Chapter 2

Physical Architectures: Interconnection and Communication

Fabrizio Petrini, Marco Vanneschi
Dipartimento di Informatica
Universitá di Pisa

Interconnection networks and routing strategies are important issues for the realisation of the physical machine in parallel computers. This Chapter deals with design methodologies and techniques, as well as with performance analysis at the physical machine level.

Section 2.1 deals with the topological properties of interconnection networks. This section discusses many properties and measures commonly used to evaluate and compare topologies and provides an overview of the properties of many widely used topologies as meshes, trees, and hypercubes.

Section 2.2 outlines the properties of the more common static topologies. The list includes rings, trees, meshes and hypercubes that are extremely popular in existing machines or research prototypes.

Dynamic interconnection networks are the subject of section 2.3.

Section 2.4 deals with the routing issues in static interconnection networks. These include setting up a path for communication, selecting a path from many alternatives, handling contention for resources, and dealing with deadlocks and livelocks. This section provides a brief overview of the many routing algorithms and techniques proposed for static networks.

The performance of an interconnection network is a critical factor and is affected by several parameters, Section 2.5 outlines some interesting results on the class of k-ary n-cubes highlighted by Dally [142] and Agarwal [10].

2.1 Design and analysis of interconnection networks

Interconnection networks can be broadly divided into two classes: static and dynamic. In a static network, point to point links interconnect the network nodes in some fixed topology. A regular topology as a mesh or hypercube is fairly common. These networks are also defined as *direct* networks because the interconnecting links are directly connected to the network nodes, as opposed to being switched dynamically.

Static interconnection networks are derived from graphs where nodes represent processors and edges represent dedicated links between processors.

In a dynamic network processing nodes are not directly connected and communicate through switches.

Nearly all networks in the literature provide some degree of symmetry. Such a symmetric topology has several advantages. First of all, it allows the network to be constructed from simple building blocks and expanded in a modular fashion. Second, a symmetric topology simplifies the use of simple routing algorithms. Third, it is easier to develop efficient computational algorithms for multiprocessors interconnected by a symmetric network. Finally, it makes the network easier to model and analyze: this is very important to hide the network characteristics from the algorithm specification and to allow the definition of an abstract machine.

An important advantage of many regular interconnection networks is modularity. The degree of a node in these networks either is independent of the network size, or grows very slowly with it (e.g. as a logarithmic function). This allows large networks to be built from simple building blocks.

The topology of an interconnection network determines many architectural features and affects several performance figures. Although the actual performance of a network depends on many technology and implementation factors, several topological properties can be used to evaluate and compare different solutions in a technology independent manner. A list of these properties follows below.

2.1.1 Node degree

The number of edges (links or channels) incident in a node is called the *node degree* d. In the case of unidirectional channels, the number of channels into a node is called *in degree*, and the number of channels out of a node is called *out degree*. The node degree is the sum of the two. The node degree reflects the number of I/O ports required per node, and thus the cost of a node. Therefore the node degree should be kept constant and as small as possible in order to reduce cost.

2.1.2 Symmetry and regularity

A *regular* network is defined as a network in which every node has the same degree. A *symmetric* (or *isotropic*) network is one in which the topology looks identical when viewed from every node or every edge.

The principal advantage of symmetry in a network lies in the ease of routing data. This allows all nodes to use the same routing algorithm. In addition, common bulk data movement operations as broadcasts and multicasts can be implemented easily and efficiently on symmetric topologies.

2.1.3 Distance measures

Communication between two nodes that are not directly connected must take place through other nodes in a static topology, or through switches in a dynamic one. The length of a communication path from a given source node to a given destination node is the number of links traversed along the path[1]. Most common static topologies provide multiple paths between pairs of nodes. The length of a (the) shortest path between a given pair of nodes is an important metric in the evaluation of static topologies. This is captured in the *diameter*, which is defined as the maximum length of the shortest paths between all possible pairs of nodes. The diameter represents the worst case distance that any message in the network may have to travel if routing is always along shortest paths. Although the diameter is useful in comparing two topologies with identical node degrees, it may be not always indicative of the actual performance. In practice, it is more important to measure the distance traveled by an "average" message. This is captured in the *mean internode distance* (or *average distance*) which is defined as the average of the lengths of all paths followed by messages in the network. While the diameter depends only on the network topology, the mean internode distance also depends on the distribution of the messages among the nodes. For any arbitrary message distribution, the mean internode distance can be computed as

$$\sum_{i=1}^{d} i * p(i)$$

where $p(i)$ is the fraction of all messages that travel a distance i and d is the diameter. In case of strict locality there is an upper bound b such that $p(j) = 0, \forall j \geq b$. The simplest message distribution used to compute the average distance is the *uniform distribution*, where each node sends messages to every other node with equal probability. With the uniform distribution, in a symmetric network, the mean internode distance may be computed as

$$\frac{1}{N} \sum_{i=1}^{d} i * n(i)$$

where $n(i)$ is the number of nodes at a distance i from a given node. Constant degree networks such a mesh, have both diameter and average distance proportional to $\sqrt{N}$; for the hypercubes and the multistage networks both the diameter and the average distance grow as $\log N$.

[1] Obviously, the actual path followed by a message depends upon the routing algorithm that is adopted. For the moment being we assume that the routing algorithm exploits the shortest physical path between two nodes.

Another performance metric is the *average traffic density* on each of the network links. The average message density is the mean number of messages carried by the link in a unit of time. This is useful to identify the bottleneck links that define the *network capacity* for a given communication pattern. For example in a binary tree the message density increases along a path from any leaf to the root under uniform traffic.

2.1.4 Bisection width

The *bisection width* is defined as the minimum number of edges that must be removed to partition the original graph of N vertices into two subgraphs of $N/2$ vertices each (if N is odd, the subgraphs are of size $(N+1)/2$ and $(N-1)/2$). The bisection width is also useful in estimating the area required for a VLSI implementation of the network. The bisection width of a two dimensional mesh with N nodes is $\sqrt{N}$ and that of a binary n-cube with $N = 2^n$ nodes is $N/2$. This comparison shows the dense connectivity of the hypercube in relation to the mesh.

2.2 A survey of common static topologies

Several static networks have been proposed and studied as possible candidates for multiprocessor and multicomputer interconnection. These include rings, tree, multi dimensional meshes and cubes, and many variants of these networks. Meshes and hypercubes are used in many commercial machines as the Cray T3D and the Connection Machine CM-2. In figure 2.1 some common static topologies are shown.

2.2.1 Rings and trees

Rings and trees are among the simplest static topologies. These networks have small and fixed degree (typically 2 or 3). A ring of N nodes with bidirectional links has a diameter of $N/2$ and average distance of $N/4$. Because both the average distance and the message density, with uniform traffic, increase linearly with the number of nodes, the usefulness of this topology is limited to small networks. The ring is also inherently unreliable as the failure of a single node or link increases the diameter of a factor of two, and more than one failure can disconnect the network. The IBM Token

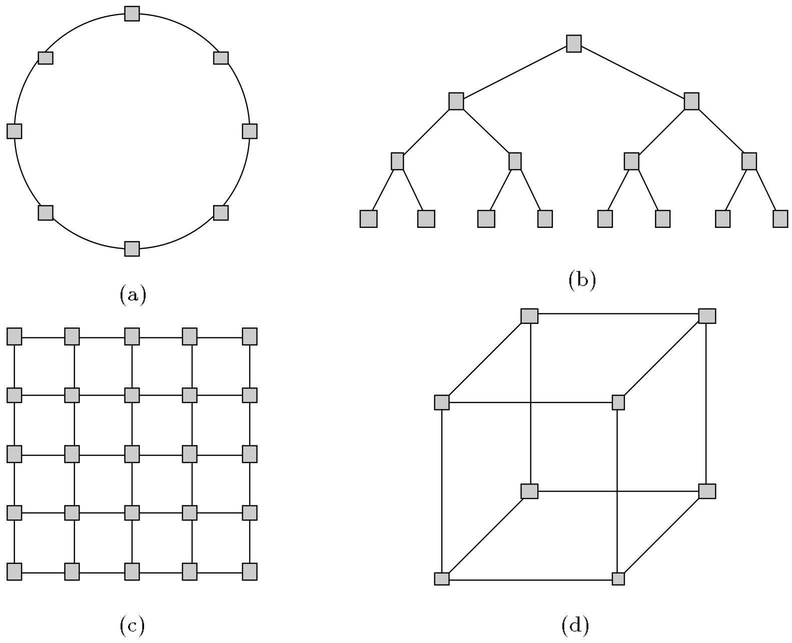


Figure 2.1: Some examples of static network topologies: (a) ring (b) binary tree (c) two dimensional mesh (d) three dimensional hypercube.

Ring and the KSR-1 (in the simple configuration without hierarchical rings) have this topology.

The topology of a binary tree is ideally matched to the communication patterns of parallel algorithms employing recursive and divide and conquer approaches. In a tree topology both the diameter and the average distance grow logarithmically with the number of nodes. The message density, however, increases more rapidly (again, assuming a uniform traffic model). In this case the message densities are highly non uniform, with the lowest densities at the leaves and the highest close to the root.

This problem gave rise to a number of approaches to reduce the contention close to the root. The *fat tree* interconnection network [296] solves the problem by progressively increasing the width of the channels from the leaves to the root (see figure 2.2).

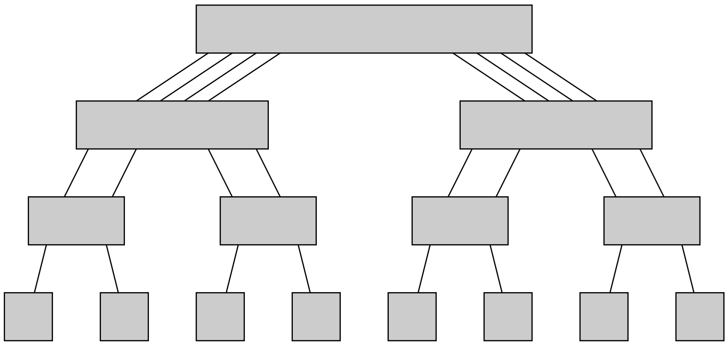


Figure 2.2: A fat tree: the number of links doubles at each level going from the leaves to the root.

2.2.2 Meshes and tori

Thanks to their simplicity and expandability, two dimensional meshes have found wide application in both commercial and experimental machines. An N nodes 2-D mesh typically consists of a $\sqrt{N} * \sqrt{N}$ grid structure with a processor at each grid point. Each of the nodes, except the ones at the borders, is connected to its four neighbors. This construction can be easily generalized to more than two dimensions. Furthermore, wraparound connections can be provided to connect nodes at the opposite edges of the same row or column. This type of topology is called toroidal mesh (see figure 2.3). Two dimensional arrays of processors are ideally suited for many applications such as image processing, matrix computations, numerical solution of differential equations, and computational geometry.

The diameter and the average internode distance of a $\sqrt{N} * \sqrt{N}$ mesh are both $O(\sqrt{N})$. Several modifications to the basic mesh topology have been proposed with the goal of reducing the diameter. These include adding a global bus or rows and columns of buses. A hexagonal mesh with six connections per node was used in the Hewlett Packard Mayfly project [157].

2.2.3 Hypercubes

The binary n-cube, commonly called the *hypercube*, has been a popular interconnection network for commercial and experimental systems, owing to

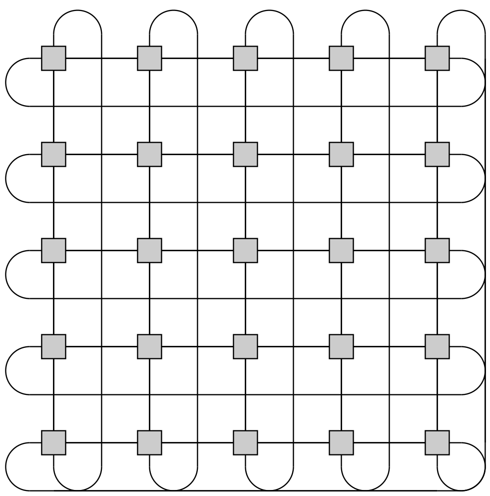


Figure 2.3: A two dimensional toroidal mesh.

its simplicity and its rich interconnection structure. The Intel iPSC/1 and iPSC/2, the nCUBE and the CM-2 were built using an hypercube structure. The hypercube graph on $N = 2^n$ nodes is obtained by labeling each node by an n-bit binary number and connecting vertices whose addresses differ exactly in one bit. Hence, the distance between two nodes along a shortest path in the hypercube is equal to the *Hamming distance* between their binary addresses. This interconnection structure allows the natural implementation of a large number of parallel algorithms, as the FFT, sorting, and neural networks. The hypercube can simulate a variety of other common topologies such as trees, rings, and meshes.

2.2.4 k-ary n-cubes

Both meshes and hypercubes belong to the general class of the *k-ary n-cube* networks. A k-ary n-cube is characterized by its dimension n and radix k, and has a total of $N = k^n$ nodes. The k^n nodes are organized in an n-dimensional mesh, with k nodes in each dimension. Figure 2.4 shows an example of k-ary n-cube. The binary hypercube is a special case of k-ary n-cube with $k = 2$. Also, the two dimensional mesh is another special case with $n = 2$. A k-ary n-cube with wraparound links is often called *torus* *connected.*

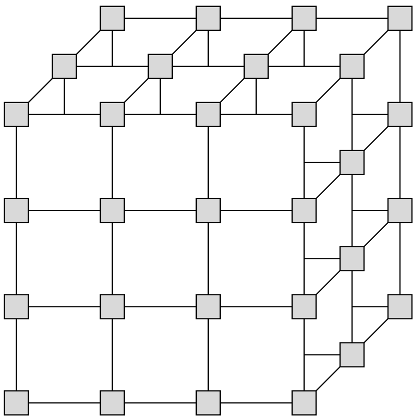


Figure 2.4: A 4-ary 3-cube without wraparound connections.

2.2.5 A comparison

The comparison of different network topologies is a difficult task, as many factors such as cost, performance, and reliability need to be considered. In addition, in the case of VLSI implementation, the amount of wiring needed for links and the number of I/O pins become important. Also, the length of the longest wire may become more important than the diameter. Some recent studies argue that meshes can perform as well, if not better, as hypercubes, in spite of their lower connectivity. One way to compensate the lower bisection width of a two dimensional mesh is to increase the width of the interconnection channels, that is to increase the bisection width in terms of the number of wires. Dally evaluated the performance of k-ary n-cubes while keeping the bisection width, in terms of the number of wires, constant. He showed that, using a wormhole flow control, a two dimensional torus provides lower average latency than higher dimensional topologies with the same number of nodes. This means that, in practice, a fair comparison has to consider the cost of all components in the implementation of these networks. Table 2.2.5 summarizes some of the properties of the static topologies.

Network	Degree	Diameter	Bisection width
ring	2	$\lfloor N/2 \rfloor$	2
binary tree	3	$2(\lceil \log_2 N \rceil - 1)$	1
$2-D$ mesh	4	$2(\sqrt{N}-1)$	$\sqrt{N}$
$2-D$ torus	4	$2\lfloor \sqrt{N}/2 \rfloor$	$2\sqrt{N}$
hypercube	$\log_2 N$	$log_2 N$	$N/2$
k-ary n-cube	$2n$	$n\lfloor k/2 \rfloor$	$2N/k$

Table 2.1: Comparison of some static topologies. All networks have N nodes and bidirectional links. The k-ary n-cube is torus connected with $N = k^n$ nodes.

2.3 Dynamic networks

Dynamic interconnection networks can be used to implement communication patterns based on program demand. Instead of using fixed connections, switches or arbiters must be used to provide dynamic connectivity. A single bus and a non blocking crossbar occupy the two ends of the performance spectrum of dynamic interconnection networks. However, to reduce costs, many networks have been designed using a large number of small crossbars organized in stages. These networks are called multistage.

2.3.1 Buses

Buses are the most common type of interconnection networks in use today's computer systems. Although a single bus allows only one pair of attached devices to communicate at any time, the performance is still adequate for many applications. Many standards for computer systems buses have evolved in the last decades. Buses have also been used to enhance the communication capability of other interconnection networks such as multidimensional meshes.

Despite of their widespread use, buses suffer from a number of limitations. The designer of a multiprocessor system often finds the single bus as the most severe obstacle to scaling up the performance of the system, either by increasing the number of nodes or by increasing the performance of existing nodes. A simple solution to the capacity problem is to use multiple shared buses.

A second problem with a single shared bus is that it introduces a single point of failure in the system, unacceptable for applications that require some degree of reliability. Again, the multiple bus approach solves this problem. A bus failure in a multiple bus system results in degraded performance, without loosing connectivity.

A third problem that imposes a limit on the number of nodes that can be attached to the bus is the fanout capability of the bus. Once again, the use of multiple buses suggests a solution to the problem by allowing each bus to be connected to only a subset of the nodes instead of the entire set. Such a multiple bus configuration is called a *partial configuration*. Some techniques for systematic design of such partial configurations are known [285].

2.3.2 Crossbars

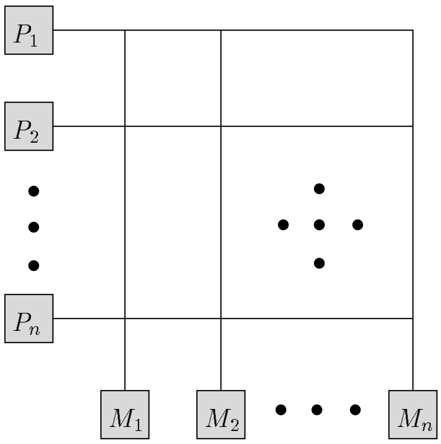


Figure 2.5: A crossbar network which connects processors to memory modules (P_i: processor, M_i: memory)

Crossbar switches allow non blocking access from any input port to any output port and therefore forms the natural building block of any low latency interconnection system. Crossbars networks have been used in computer systems for a long time, for example in the C.mmp system [450]. Crossbar networks are also the basic building blocks used to form other dynamic networks such as multistage networks. Crossbars find application even within the individual nodes of static networks such as hypercubes and meshes [121], [138], where they facilitate fast routing of data through intermediate nodes. A conventional two sided crossbar consists of a set of input lines and a set of output lines, logically placed perpendicular to each other, with switches placed at each point where the lines cross, as shown in figure 2.5. Large crossbar switches are implemented by partitioning the matrix into smaller square boxes and assigning each block to a chip. Crossbar networks have not found wide use in large multiprocessors systems because of two primary reasons:

- A crossbar requires $O(N^2)$ gates to interconnect N ports, as compared to $(N \log_2 N)$ needed for a multistage network (see section 2.3.3 below). Thus, the circuit complexity of large networks is still considered beyond the capability of current technologies.

- The number of pins on an integrated circuit package does not permit large crossbar networks to be integrated on a single chip.

- Although the hardware complexity of $O(N^2)$ for a full crossbar with N ports imposes a limit on their size, crossbar networks are still attractive for interconnecting a moderate number of modules. With a crossbar network, N processors can simultaneously access data from N memory modules and any degradation in performance occurs only for memory conflicts or external conflicts.

2.3.3 Multistage networks

Many multistage interconnection networks have been proposed for multiprocessors systems over the years. These include the omega, the butterfly, the baseline, the delta, the banyan and a number of variants and derivatives [295]. These networks use fewer gates than the crossbar network and are typically constructed for $N = 2^n$ inputs and outputs with multiple "stages" of switching elements. A basic requirement in the design of these networks is to provide "full access" capability, which means that any input terminal of the network should be able to access any output terminal in one sweep through the network. Most of the multistage networks can be constructed with $2 * 2$ crossbars switches as basic elements, and have $n = \log_2 N$ switching stages with each stage consisting of $N/2$ elements. Thus the total number of gates in these networks is $O(N \log N)$, as compared to $O(N^2)$ for a crossbar. These networks can also be constructed with larger switching elements, and fewer stages.

2.3.4 Butterfly networks

The n-dimensional butterfly has $(n+1)2^n$ nodes and $n2^{n+1}$ edges. The nodes correspond to pairs $\langle w, l\rangle$, where l is the *level* of the node $(0 \leq l \leq n)$, and w is a n bit binary number that denotes the row of the node. Two nodes $\langle w, l\rangle$ and $\langle w', l'\rangle$ are linked by an edge if and only if $l' = l + 1$ and either

- w and w' are identical or
- w and w' differ in the l'-th bit only.

If w and w' are identical, the edge is said *straight edge*. Otherwise, the edge id a *cross edge*. In addition, edges connecting nodes on levels l and $l+1$ are said to be *level* $l+1$ *edges*. The butterfly and the hypercube are quite similar in structure. In particular the l-th node of the n-dimensional hypercube corresponds to the l-th row of the n-dimensional butterfly (i.e. we can obtain a hypercube from a butterfly by merging all butterfly nodes that are in the same row and then removing the extra copy of each edge). Because of the great similarity between the butterfly and the hypercube, these topologies share some nice properties. First they have a simple recursive structure. For example, it can be seen from figure 2.6(a) that one n-dimensional butterfly contains two $(n-1)$-dimensional butterflies as subgraphs (just removing the level 0 nodes; alternatively, we could remove the level n nodes, as is done in figure 2.6(b)). Another useful property of the n-dimensional butterfly is that each level 0 node is connected to any level n node by a unique path of length n. The path traverses each level exactly once, using the cross edge from level l to level $l+1$ if and only if w and w' differ in the $(l+1)$-bit only. For example, see figure 2.7. As a consequence, we can have just this routing algorithm in this topology, and the n-butterfly with N nodes has diameter $O(\log N)$. Like the hypercube, the butterfly has a large bisection width of $\Theta(N/\log N)$.

We can generalize the butterfly topology to the class of k-ary n-butterflies. The n-dimensional k-butterfly consists of $(n+1)k^n$ nodes of the form $\langle w, l\rangle$, where $w \in \{0, 1, \ldots, k^n - 1\}$ and $l \in \{0, 1, \ldots n\}$. For example a 4-ary 2-fly is shown in figure 2.8. This topology is widely used in architectures whose switches have degree greater than 4 [336].

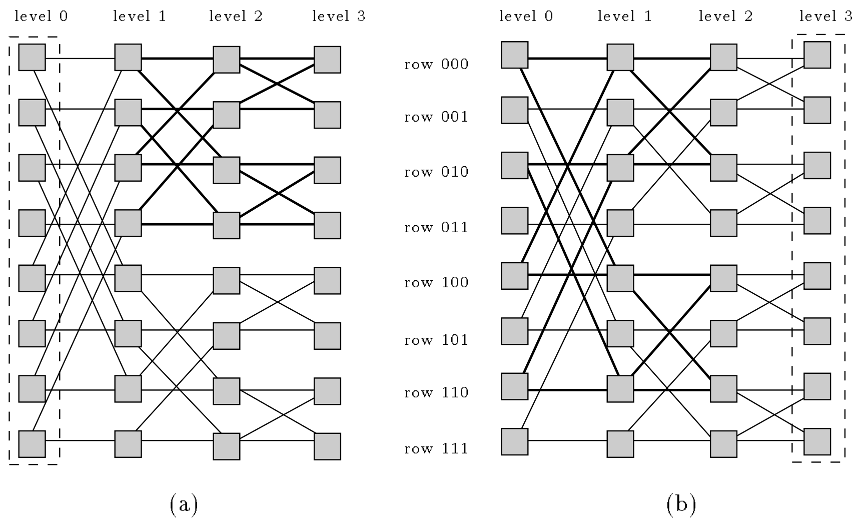


Figure 2.6: (a) A three dimensional butterfly contains two butterflies of dimension two, that can be obtained removing the level 0 nodes. (b) Alternatively, removing the level 3 nodes we can have two disjoint butterflies, one consisting of the even rows, and one of odd rows.

2.3.5 Generalized fat trees

We already described the fat tree topology in the class of direct topologies. A commonly used topology in existing commercial machines [297], which is quoted as "fat tree", can be considered as a generalization of the butterfly topology shown above. This is build out of a set of crossbar switches and processing nodes, as shown in figure 2.9. It is easy to recognize a butterfly topology whose level n nodes are connected to a set of processing nodes. For this reason the generalized fat trees share many properties with the butterfly. While a message is forced to cross the whole butterfly, the generalized fat tree allows the exploitation of message locality and clustering. A message typically moves upward until it gets to a *nearest common ancestor* (NCA) of both sender and receiver. Then goes down towards the destination using the only shortest path available. It is worth noting that not all the paths are available at the NCA level, due to the use of limited size crossbars. As an example, incoming messages at the switch SW from the

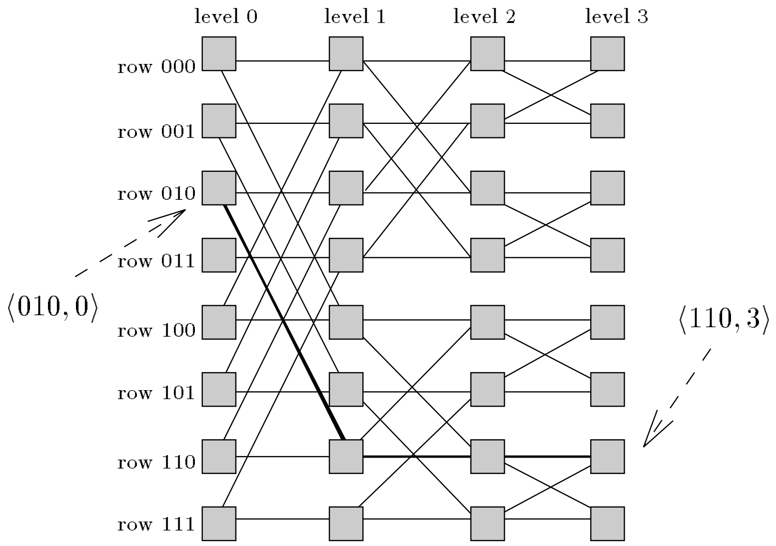


Figure 2.7: The unique path of length 3 from $\langle 010, 0\rangle$ to $\langle 110, 3\rangle$. Each level is traversed once.

link l_1 of the figure 2.10, can use only the outgoing links l_1 and l_3, even if the logical structure of the fat tree allows more choices, because each switch may be decomposed into a set of limited size switches, each one connected to a subset of the input/output links.

The generalized fat tree allows an easy implementation of randomized and adaptive deadlock free routing algorithms. A randomized algorithm can be implemented choosing one of the available NCA randomly in the ascending phase, and then resorting to deterministic routing in the descending phase. This algorithm has the nice property that the average path length is unchanged with respect to a deterministic version. In the adaptive routing, choices in the ascending phase are done according to the local condition of the switch where the routing decision is taken. This is simpler than the routing algorithms for the k-ary n-cubes (in particular the adaptive ones) outlined in the next section. However, the wiring problems of the generalized fat tree and the increasing clock rates may favor the utilization of low dimensional cubes in the physical implementation of

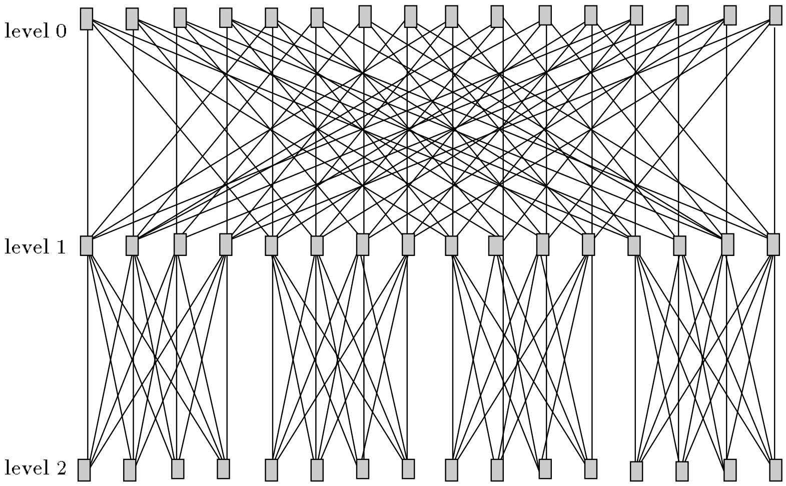


Figure 2.8: A 4-ary 2-fly.

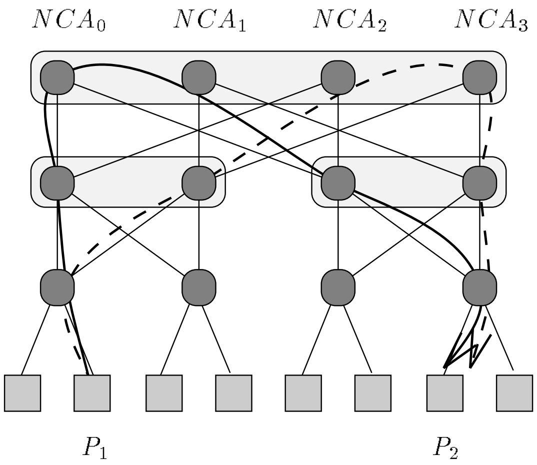


Figure 2.9: A generalized fat tree is built out of a collection of fixed degree switching elements. A message can be sent from processor P_1 to processor P_2 using one of the four available paths passing through the nearest common ancestors.

an interconnection structure. Generalized fat trees are still very interesting for the design of logical interconnection networks that can be embedded in a physical topology, using the virtual channels technique.

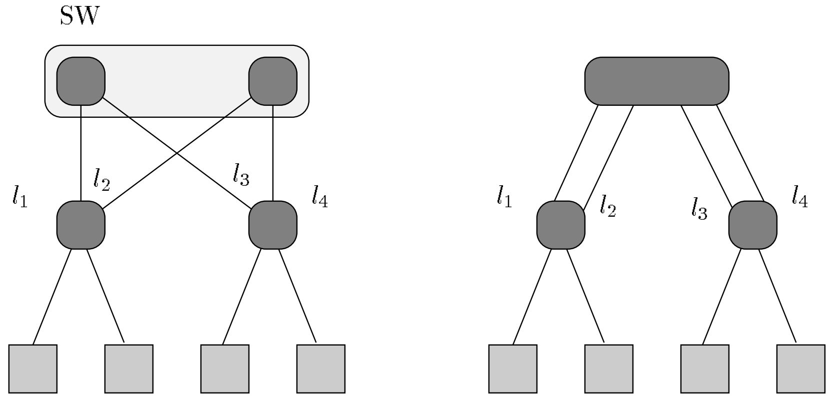


Figure 2.10: Incoming messages from link l_1 can only use outgoing links l_1 and l_3 on the switch SW in the generalized fat tree on the left-hand side. In the corresponding fat tree on the right-hand side, all links $l_1, \ldots, l_4$ can be used instead.

2.4 Routing strategies

The routing strategies employed in an interconnection network can be classified along different dimensions. Kung [280] introduced a taxonomy of these approaches. In figure 2.11 some of these dimensions are shown together with a classification of the more common solutions.

2.4.1 Path setup

The path between two communicating nodes can be setup *statically* or *dynamically*. In the static approach, the path is configured before execution of a computational task and stays in effect for the duration of the program. Such an approach can be used only when the communication pattern is known in advance, and when enough resources exist in the network for configuration of dedicated paths. For example, the static approach is adequate

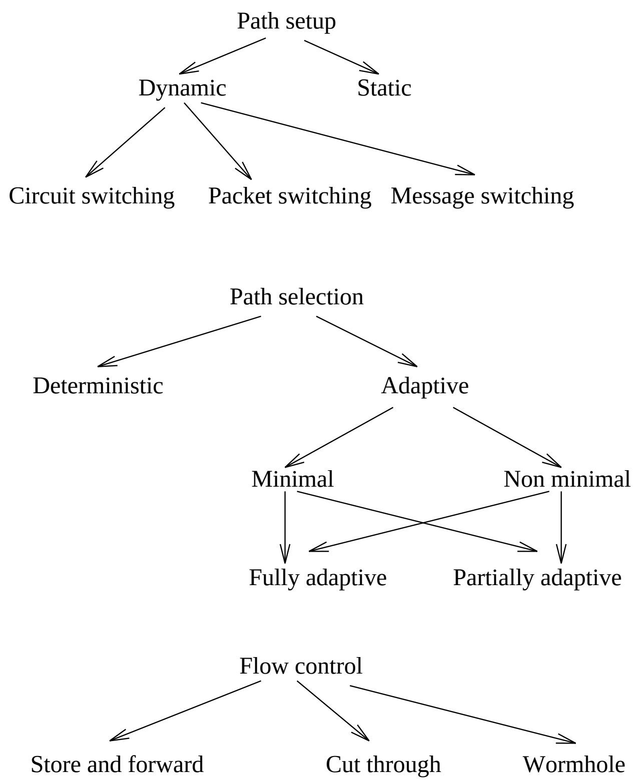


Figure 2.11: A taxonomy of communication methods employed in interconnection networks.

if the computational algorithm has been compiled to the network such that communications occur between nodes that are directly connected [369].

In most cases, however, more general communication patterns must be supported, because the communication needs cannot be predicted in advance. This requires path setup to be performed dynamically on demand. There are some well known approaches to dynamic path setup.

- In *circuit switching* communication proceeds in two phases: a path setup phase followed by the data transfer phase. This has the advantage that no buffers are needed along the path. Also, all data move along the same path. The main disadvantage is the overhead for the path setup, that can be justified only by long messages.

- *Packet switching* eliminates this overhead by allowing data to be sent without an explicit path setup phase. The message is formatted in packets, that are sent from node to node until they reach the destination. If a packet finds a busy link along its path, it must be blocked, queued, discarded or misrouted, depending on the flow control method in use. This results in non deterministic delays. A large message is typically split into a collection of packets, and each packet is sent independently. The destination then re-assembles the packets into the original message.

- In *message switching*, the messages generated by the user program are not broken into smaller packets as in message switching, but transmitted as a single unit. This avoids the re-assembly problem, but may increase the amount of buffer memory needed at each node.

2.4.2 Routing path selection

Most static interconnection networks provide multiple physical paths for routing a message between two given nodes. This introduces the problem of choosing a route between many alternatives.

The routing path selection can be characterized by a relation R and a function ρ, defined on the set of physical links L and the set of nodes N:

$$R \subset L \times N \times L,$$

$$\rho : P(L) \times \alpha \mapsto L.$$

The routing relation R identifies the permissible paths that a message can use to reach its destination. Given the current link $l \in L$, and the destination node $n \in N$, R identifies a set of permissible links $P(L) \subset L$ that can be used in the next step on the route. R is a relation because there can be several alternative paths to get to the destination.

The function ρ selects one link from the set of permissible ones. At each step of the route, ρ takes the set of available links $P(L)$, some additional information on the state of the network α, and picks a particular link $l \in L$.

The simplest approach is the *deterministic* one, where the route is fully determined by the source and destination addresses. This has the advantage of being very simple, but is unable to adapt the network to conditions such as congestion or failures. *Adaptive routing* is very important to provide a network performance which is less sensitive to the communication pattern. In this case, the paths can be chosen according to the degree of congestion of the node where the routing decision is taken.

In *deterministic* routing the communication path between a given source node and a given destination node is fixed by their addresses. This forces communication to take place on a unique path, even when multiple paths are available. A common example of fixed path routing is the *row-column* or *Manhattan* routing in the two dimensional mesh, which always route a message along the row first, until alignment is reached with the destination, and then along the column. Another common example is the *e-cube* routing for the hypercubes. This algorithm routes the packet in increasing or decreasing order of dimension. Consider a source node $S = s_1, s_2 \ldots s_n$ and a destination node $D = d_1, d_2 \ldots d_n$. If messages are routed in increasing bit order, the message from source S moves from node to node by correcting the bits of S from the least significant bit to most significant one, to match the bits of the final destination D.

Deterministic routing is easy to implement but lacks the flexibility to adapt to network conditions as congestion or failure. In the two dimensional mesh shown in figure 2.12(a), if all the nodes in the last row send a message to a distinct node in the last column, all the messages cross the "hot" node in the lower right corner, even if there are many alternative paths to handle the suggested communication pattern. An adaptive routing can route these messages along disjoint paths, as shown in figure 2.12(b). A similar problem arises in the presence of faulty links.

A *minimal* adaptive routing algorithm limits the path selection to the shortest paths between any given pair $\langle source, destination \rangle$. With a *non minimal* routing algorithm, the selected path may not always be a shortest path.

A minimal or non minimal adaptive routing can be either *fully adaptive* or *partially adaptive*. A fully adaptive routing can select any path from the set of the available ones (this is the set of the shortest paths in case of a minimal algorithm). A partially adaptive algorithm limits the selection to a proper subset. A fully adaptive algorithm is often difficult to implement and can easily generate deadlocks. In practice, a partially adaptive algorithm may still have enough degrees of freedom in selecting paths.

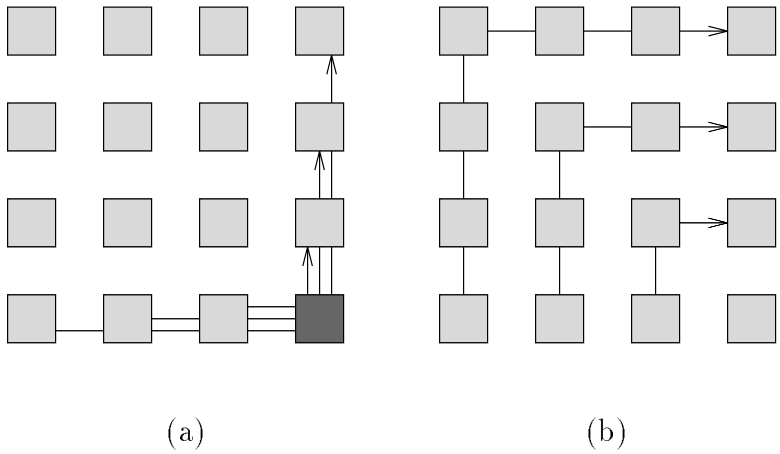


Figure 2.12: (a) A set of routing paths created by a deterministic routing algorithm in a two dimensional mesh. (b) The same pattern routed adaptively to create disjoint paths.

2.4.3 Flow control

When communication is performed without waiting for the path setup to be complete, as in packet switching, a packet may block within the network as a result of network congestion. Flow control has to regulate the movement of packets from node to node so as to utilize the network resources, buffers and communication channels, as efficiently as possible.

- A simple approach to implement network flow control is *store and forward*. In this method, the entire packet (message) is buffered at each intermediate node and forwarded to the next node in its path as soon as the desired outgoing link becomes available. This switching strategy was adopted in early prototypes and commercial machines including the iPSC/1, nCUBE-1 and those based on the Transputers T414 and T800. Store and forward flow control is simple but has some drawbacks. First, each node must buffer every incoming packet, using memory space. Second (and more important), the network latency is proportional to the distance between the source and destination nodes. The network latency is, in the absence of contention, $(L/B)D$, where L is the packet length, B is the channel bandwidth, and D is the length of the path between the two nodes.

- An improvement over the store and forward approach, called *virtual cut through*, avoids this problem by buffering a packet only when it encounters a busy link [268]. Cut through routing was used in the torus routing chip implemented at Caltech [148].

- Buffering can be reduced to a minimum if the blocked packet can stay in multiple nodes along the path, holding the links between them. The *wormhole* approach is based on this idea [149]. A packet is divided in *flits* (flow control digits) for transmission. The header flit forces the route. As the header advances, the remaining flits follow in a pipeline fashion. If the header finds a link already in use, it is blocked until the link becomes available. The network latency for wormhole, again, in the absence of contention, is $(L_f/B)D + L/B$, where L_f is the length of each flit, B is the channel bandwidth, and L is the length of the message. If $L \gg L_f$, the path length will not significantly affect the network latency unless D it is very large. For low to moderate traffic, wormhole flow control outperforms store and forward. On the other hand, wormhole can easily introduce deadlocks by causing cyclic dependencies in the resources allocation if the path selection and the link allocation are not performed carefully. The first prototype that adopted wormhole flow control was the Ametek 2010, which used a two dimensional mesh topology. The Intel Touchstone Delta and Intel Paragon, the MIT's J-machine, the CMU iWarp, and the Transputer T9000 family use wormhole in their communication building blocks too.

2.4.4 Deadlock avoidance

An important issue that must be addressed in the design of a routing algorithm is the possibility of deadlock. Deadlocks occur as a result of cyclic waits for resources, such as buffers or links. Various approaches can be used to deal with deadlock. A preemptive approach can be used to break deadlock once it has occurred. This option is not very popular because it is not easy to detect deadlock in a distributed environment. The most favored approach is to prevent deadlocks by careful design of the routing algorithm. The basic idea, common to several approaches, is to constrain the order in which resources are allocated, to prevent cyclic wait situations. This can be achieved:

- By partitioning the available buffers in a node into multiple classes and controlling the allocation of packets to classes. This approach is widely used in packet switched, store and forward routing.

- By multiplexing the capacity of a physical link among multiple virtual channels and controlling their allocation to communication requests. This technique has been successfully used with wormhole flow control. Through virtual channels, a physical network can be partitioned into disjoint logical networks, facilitating the implementation of adaptive routing algorithms. Also, virtual channels are useful for many other reasons. By increasing the logical degree of connectivity in the network they facilitate the mapping of applications in which processes communicate according to another logical topology. Furthermore, they provide the ability to deliver guaranteed communication bandwidth to certain classes of packets. For example, it may be important to reserve some bandwidth to support higher priority messages or system related functions, such as debugging or monitoring.

- By constraining the path selection process. This approach involves designing the routing algorithm such that cyclic wait situations are avoided. The technique is often coupled with the virtual channels. Many such deadlock free algorithms for the k-ary n-cubes have been proposed in the literature.

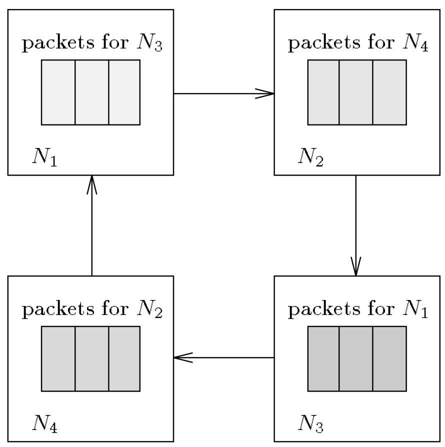


Figure 2.13: A store and forward deadlock.

Store and forward deadlocks occur when the buffers in a set S of nodes become full with packets addressed to nodes belonging to S, but none of the packets can advance. Figure 2.13 illustrates a store and forward deadlock where all the buffers of node N_1 store packets destined to node N_3, all the buffers in N_2 store packets for N_4, and so on. Several proposed approaches to prevent store and forward deadlocks are based on partitioning the set of buffers in each node into disjoint classes, and enabling them to packets that satisfy particular conditions. These are known as *buffer reservation schemes*. Our scheme allocates the buffers on the base of the number of hops a packet has traveled. That is, a packet arriving at a node after i hops is stored in a buffer belonging to class i. This requires at least $h+1$ distinct buffer classes in a node, where h is the maximum number of hops that any packet can travel (this is the same as the diameter in case of minimal routing). The existence of deadlock in a store and forward network can be detected building the *buffer dependency graph*, corresponding to the buffer reservation scheme in use. The buffer dependency graph is a directed graph built as follows. Each processing node n is represented as b vertices $v_1(n) \dots v_b(n)$, where b is the number of buffer classes for each node[2]. An edge is inserted between two vertices $v_i(n)$ and $v_j(k)$, $n \neq k$, belonging to distinct nodes, if the adopted algorithm can route a packet from $v_i(n)$ to $v_j(k)$. For example, figure 2.14 shows a 4 node network using the buffer reservation scheme discussed above. In [222] is shown that a store and forward deadlock can occur only if the buffer dependency graph has a directed cycle.

Just as the problem of deadlock in store and forward routing is solved by dividing the set of buffers in classes, the mostly used approach to deadlock avoidance in wormhole routing consists in splitting the physical channels into virtual channels. This is obtained by multiplexing a number of virtual channels onto the same physical channel on demand. Each virtual channel is provided with its own data structures. Deadlock is prevented by constraining the allocation of the virtual channels. As in the buffer dependency graph, a *channel dependency graph* can be used to find out deadlocks for a given routing algorithm. In the channel dependency graph each vertex corresponds to a virtual channel, and a directed edge is inserted between two vertices i and j if and only if the routing algorithm allows a message arriving on the channel corresponding to i to be routed to the channel corresponding to j. To determine the existence of the edge $\langle i,j \rangle$, we notice that, with the introduction of virtual channels, the routing relation R and the function ρ can be expressed in terms of virtual channels rather than physical links. If we denote with $R_C \subset C \times N \times C$ the routing relation on the set of virtual channels C, then the edge $\langle i,j \rangle$ exists iff there is a node n such that $(i,n,j) \in R_C$. The channel dependency relation $dep \subset C \times C$

[2]Each class can have one or more buffer positions.

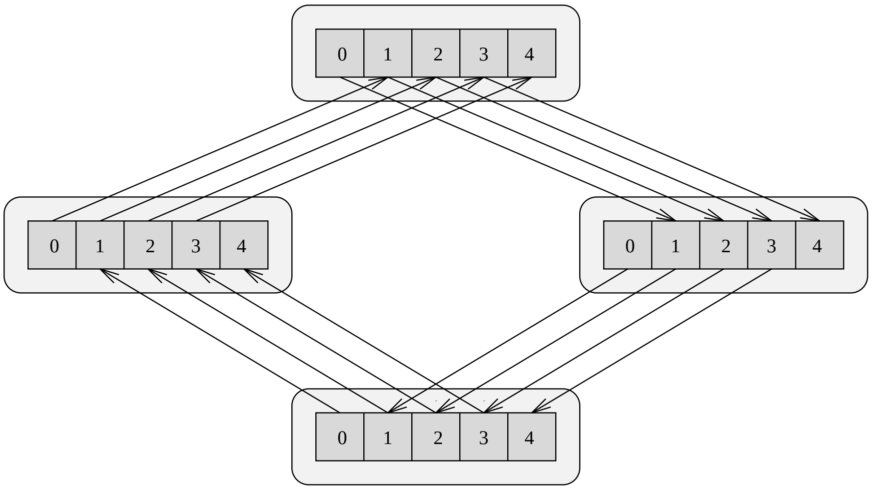


Figure 2.14: A four node network that uses the hop based reservation scheme. The arrows show the edges of the associated buffer reservation graph.

is a projection of the routing relation R_C onto $C \times C$. A pair of channels (i, j) is in *dep* iff $\exists\ n \in N$ such that $(i, n, j) \in R_C$. Dally and Seitz [149] found that deadlocks can occur for a given routing algorithm if the corresponding channel dependency graph has a directed cycle. By consequence, the algorithm is deadlock free if the graph has no cycles.

An example of application of the channel dependency graph is shown in figure 2.15. Routing in the ring (4-ary 1-cube) of figure 2.15(a) is deadlock prone, because its channel dependency graph contains a cycle, as shown in 2.15(b).

Each physical channel c_i can be associated with two logical channels c_i^H e c_i^L (figure 2.15(c)), with the following routing relations in R_C:

$$(c_2^L, N_3, c_3^L),\quad (c_3^L, N_4, c_4^L),\quad (c_4^L, N_1, c_1^H)$$
$$(c_1^H, N_2, c_2^H),\quad (c_2^H, N_3, c_3^H).$$

R_C corresponds to the simple routing algorithm that sends a message from source node N_i to the destination N_j

- using high channels $c_s^H,\ s \in \{i, \ldots, j-1\}$ if $i < j$;

- using low channels c_t^L $t \in \{i, \ldots, 4\}$ and than switching to high channels c_u^H $u \in \{1, \ldots j-1\}$, otherwise.

The channel dependency graph shown in figure 2.15(d) [3] converts the cycle into a spiral, providing a total order between channels.

The correspondence between virtual channels in wormhole routing and buffer classes in store and forward is shown in [77]. The authors point out that, for every deadlock free buffer dependency graph for a store and forward network, it is possible to design a corresponding deadlock free routing algorithm that uses virtual channels.

2.4.5 Adaptive routing algorithms

Several approaches to implement adaptivity in store and forward interconnection networks are based on the *deflection routing*[4]. Deflection routing was first proposed many years ago [55]. Deflection routing attempts to solve the problem of long waiting times in case of blocking by misrouting a packet on any available link when it cannot be transmitted on the requested outgoing link. This means that packets may be routed to wrong intermediate nodes and thus take longer (non minimal) paths. The additional delay due to the longer path may be offset by the absence of queuing delays at the nodes. Deflection routing algorithms are particularly attractive for fixed degree networks. Consider a two dimensional torus with bidirectional channels: assuming synchronous operation and fixed size packets, a node receives at most four packets in a single sweep. If several packets request the same outgoing link, all but one of the packets are deflected and transmitted on idle links. It is worth noting that in a processing node no buffering is needed, but for the local outgoing and incoming messages. Deflection routing introduces the problem of livelocks, caused by packets that may never get to their destination. The most common solution is to associate packets with deflection or aging counters, that are incremented every time the packet is misrouted. Priority is given, in case of conflict, to "older" packets. This type of routing is used in the CM-2 and in the Tera Computer.

Many wormhole adaptive algorithms have been developed for interconnection networks in the k-ary n-cube family. These algorithms display interesting tradeoffs between their degree of adaptivity and the number

[3] Dotted circles in figure 2.15(d) represent transitions that are not used by the routing algorithm.

[4] Also called "hot potato routing" or misrouting.

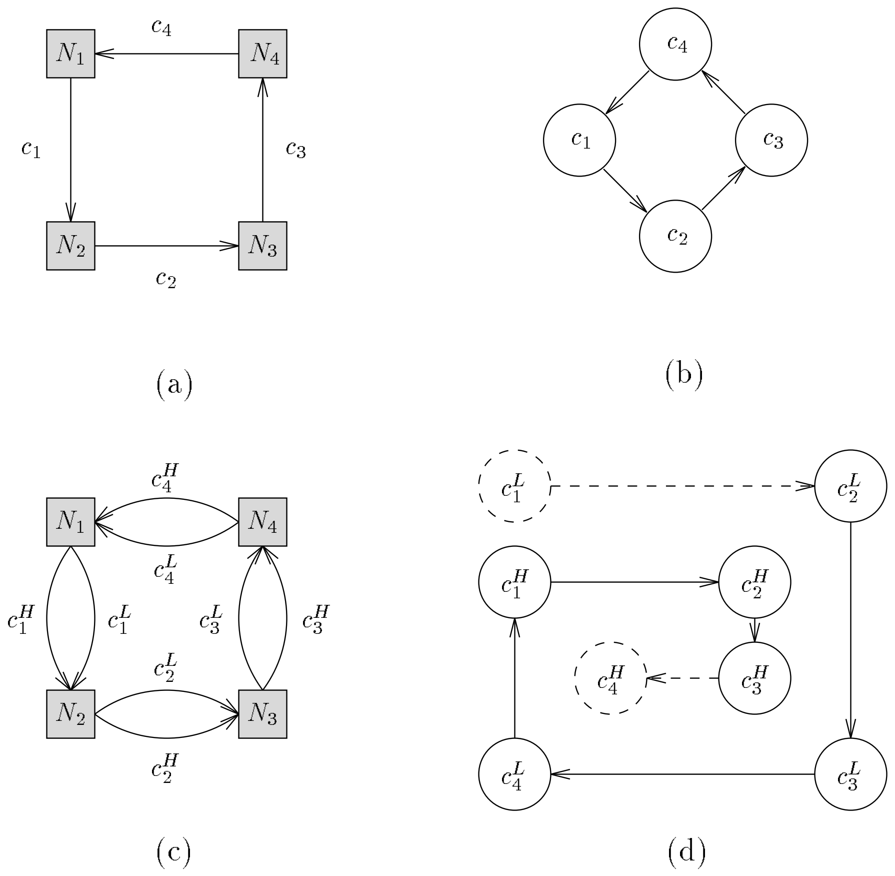
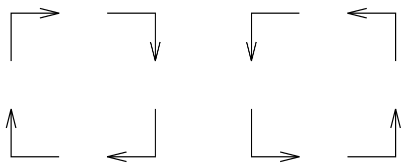


Figure 2.15: (a) A ring with unidirectional channels. (b) The associated channel dependency graph contains a cycle. (c) Each physical channel c_i is logically split into two virtual channels c_i^H e c_i^L. (d) A modified channel dependency graph without cycles.

of virtual channels needed. If the dimensions are routed strictly in order, as in row column or e-cube routing, no deadlocks can result in a mesh and binary n-cube, even with one virtual channel for physical channel. Introducing adaptivity usually requires an increase in the number of virtual channels. For example, Linder and Harden [303] describe fully adaptive minimal algorithms for k-ary n-cubes with unidirectional and bidirectional links. For the unidirectional case, $(n+1)$ virtual channels are needed for physical channel. For k-ary n-cubes with bidirectional links, 2^{n-1} virtual channels are needed per physical link in each direction, if the network has no wrap around links (mesh connected). With wrap around links, the number increases to $(n+1)2^{n-1}$.

Chien and Kim [110] presented an approach to trade off adaptivity against the number of virtual channels. Their algorithm, called *planar adaptive routing*, is minimal and partially adaptive. This approach involves examining the routing dimensions in pairs, and constraining the routing choices at any time to one or two dimensions. This, in general, is less flexible than the fully adaptive algorithm of Linder and Harden, but requires only a constant number of virtual channels, regardless of the network dimension. For example, in a k-ary n-cube without wrap around connections, only three virtual channels for each physical link are required.

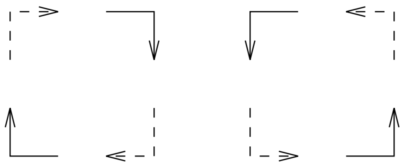


(a) (b)

Figure 2.16: The eight possible 90 degree turns in a routing path in a two dimensional mesh.

Stricker [418] studied deadlock free adaptive routing in two dimensional meshes and observed that deadlocks occur when packets change their direction in the network. Assuming no backtracking and only one virtual channel for each direction, such a change of direction in a two dimensional mesh is always a 90° turn. Labeling the four directions as north, south, west and east, the possible turns a packet can take are illustrated in figure 2.16. There are eight different turns; for the deadlocks shown in figure 2.16(a) and 2.16(b) to occur, the packets involved must have taken all the four turns. Thus, prohibiting just one of the turns in each of the figures is sufficient to prevent deadlock.

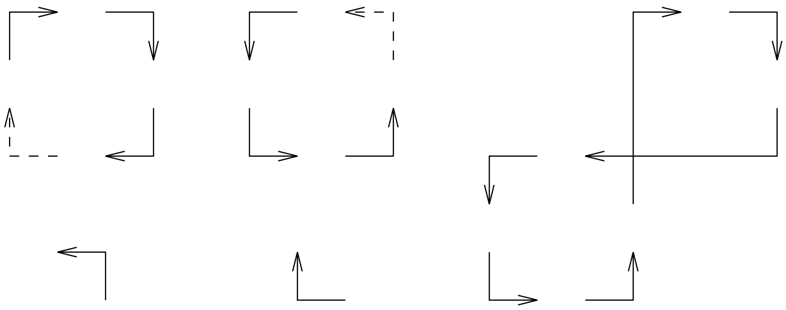
The row column routing prevents deadlock by prohibiting four of the turns (figure 2.17). The remaining four turns cannot form a cycle; they also do not allow any adaptiveness. As shown above, deadlock can be prevented by eliminating less than four turns. In fact, only two turns need to be



(a) (b)

Figure 2.17: Only four turns (solid lines) are allowed in the row column algorithm.

prohibited, one for each abstract cycle, allowing for some adaptiveness. Prohibiting any two turns will not prevent deadlock, as shown in figure 2.18(c). The three right turns allowed in figure 2.18(a) are equivalent to the prohibited left turn in figure 2.18(b), and, symmetrically, the three left turns allowed in figure 2.18(b) are equivalent to the prohibited right turn in figure 2.18(a). Thus, both cycles exist and deadlock is still possible.



(a) (b) (c)

Figure 2.18: Six turns that complete the abstract cycles and allow deadlock.

The *turn model* [213] prevents some of the transitions between dimensions, and generalizes to multidimensional meshes and binary n-cubes. These schemes require only a virtual channel per physical channel, are non minimal, and partially adaptive. The approach for two dimensional meshes works by disallowing two of the eight possible turns a packet may take.

Of the sixteen different ways to prohibit these two turns, twelve prevent deadlock (see figure 2.18(c) for a deadlock case), and three are unique if symmetry is taken into account. For example, disallowing the north to west and east to south turns, results in the *negative first* routing algorithm. This algorithm implements routing in two phases: in the first phase routing takes place in negative directions only (south and west), and the second phase allows routing in the positive directions only (north and east). Figure 2.19 shows some examples of negative first paths on a two dimensional mesh.

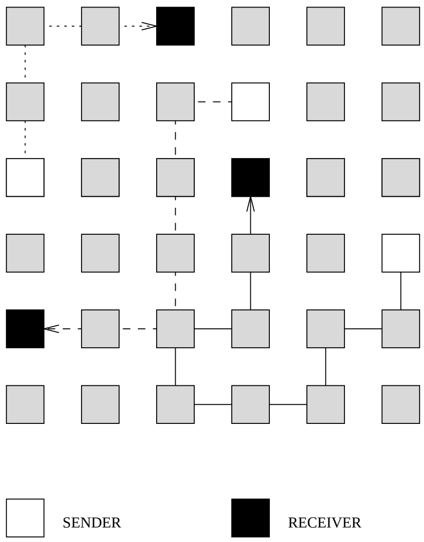


Figure 2.19: Examples of the negative first algorithm.

This algorithm can be easily generalized to k-ary n-cubes without wrap around links. An n-dimensional mesh can produce $n(n-1)$ distinct cycles, each involving four 90° turns. To prevent deadlock one turn in each of the $n(n-1)$ cycles is not allowed. Generalization to the binary n-cube results in the *p-cube algorithm*. The algorithm divides the dimensions to be routed into two sets. One consists of the negative hops, that is the set of dimensions where the source address has a 1 in its binary representation and the destination has a 0. The other consists of the positive hops. As in the case of meshes, the p-cube algorithm consists of two phases. The non minimal version of the algorithm increases the routing choices in the first phase by allowing routing also along the dimensions in which both the source and the destination addresses have a 1-bit; the second phase is unchanged. In both minimal and non minimal versions, the dimensions can be routed in any order during each phase (but all negative hops must be completed before the positive ones).

The turn model can be used in conjunction with virtual channels to increase adaptivity and to generalize to torus connected k-ary n-cubes. Thus, instead of prohibiting some turns, the packet can be switched to a different virtual channel upon taking such a turn. Dally and Aoki [146] used this idea to design partially adaptive non minimal routing algorithms for the class of k-ary n-cubes. In their algorithms, each packet carries with it a dimension reversal number[5], which keeps track of the number of times the packet has been routed from a channel in one dimension p to a channel in a lower dimension $q < p$.

The *dimension reversal* algorithms partition the virtual channels of each link into τ classes. After being routed on a virtual channel of a class $i \in \{0, \ldots, \tau-1\}$, a message cannot be routed through a channel belonging to a lower order class.

In the *static dimension reversal* algorithm, τ is equal to the maximum number dimension reversals experienced by a message, as recorded by the dimension reversal number on each packet.

In the *dynamic dimension reversal* $\tau = 2$ and the channels in class 0 and 1 are called, respectively, adaptive and deterministic. First packets are routed along adaptive channels, in any direction and without constraining the number of dimension reversals, provided that a packet with dimension reversal p does not wait on a channel occupied by a packet with dimension reversal q, if $q \leq p$. If a packet cannot be routed along any adaptive channel, it switches to deterministic channels. Since packets are more likely to be routed along deterministic channels in the presence of congestion, these algorithms (in particular the dynamic one) collapse when a large number of packets is injected into the network in a short time.

To solve this problem, non monotonic allocation strategies have been proposed [174] [63]. These algorithms assume $\tau = 2$, as in the dynamic reversal case, but a message can be routed along adaptive channels after using deterministic ones. Experimental results [197] show that these algorithms are less sensitive to congestion and can reach a performance close to the network capacity.

[5] Initialized with 0.

2.5 Performance evaluation

The design of an interconnection network for a multiprocessor or a multicomputer inevitably involves tradeoffs between performance and cost and physical constraints. The goal of the design is to achieve the highest performance at a given cost, or to minimize the cost given some performance constraints.

Performance modeling can be done analytically or by simulation. A simulation model can provide details that are difficult to model analytically, but can be time consuming to develop, execute, and validate.

The type and the parameters of the traffic distributions used in the performance analysis have a significant influence on the results. A difficult task is to determine proper distributions of traffic that are representative of real applications, and do not make the analysis too complex. In the absence of such a characterization, the traffic distribution has been chosen to simplify the resulting model. While these simplified traffic models limit the utility of the results obtained, they still provide insight into the behavior on the network and the influence of certain parameters on the overall performance.

Roughly speaking, the traffic distribution has two dimensions, space and time. The spatial dimension describes how the traffic originating at a node is distributed among the possible destinations. The temporal dimension describes how traffic varies with time. The requests originating at each node are represented as a two dimensional random process. In practice, a number of simplifications is usually made. Usually the spatial distribution is time invariant; most studies also assume that the traffic generation at every node is controlled by the same statistical process, as a Poisson distribution. Because most interconnection networks have a regular topology, these assumptions bring symmetry to the traffic at various points in the network, simplifying the modeling task considerably. The simplest model for spatial distribution is the uniform traffic model, which treats each of the destinations equally likely. In a multicomputer with N modes this implies that a message generated at a particular node is addressed to one of the remaining nodes with equal probability.

The commonly used performance metrics for packet switched interconnection networks are *latency* and *throughput*. The packets are subject at queuing at intermediate nodes and suffer from non deterministic delays. The average latency of a packet from its source to its destination is therefore an important measure of the performance of the network. Throughput is the average amount of data delivered in the unit of time: this metric provides an interesting measure of the network as a "whole".

2.5.1 Performance under physical constraints

In high performance parallel machines the performance of the interconnection networks is increasingly limited by physical constraints. The speed of the light limits the length of wires. Packaging technology limits the number of external connections or pins in a given processor. Wire density limits the physical embedding of a logical topology into the three dimensional space.

Dally [142] and Agarwal [10] extensively analyzed the class of k-ary n-cubes under these constraints. The latency through a network without considering contention[6], T_b, is simply the product of the time spent in a node with the sum of the nodes and the message length, when wormhole flow control is used. The time spent in a node is the sum of the switch and wire delay. For a torus connected k-ary n-cube with unidirectional channels the base network latency can be approximated as [10]

$$(s + k^{n/2-1}) \left(n\frac{k-1}{2} + \frac{L}{W} \right),$$

where

- n and k ($N = k^n$) are the network dimension and radix,
- s is the switch delay,
- L and W are, respectively, the message length and the channel size in bits.

This formula takes into account the wire delay normalized with respect to the wire delay of a two dimensional cube (assuming a linear model for the wire delay).

Figure 2.20 shows the base network latency as a function of the network dimension. Constraints on the bisection width and the pin count are not considered (this allows nodes with an unbounded number of external connections for higher dimensional networks). The plot focuses on the limits of signal propagation alone. This physical limitation tends to favor low dimensional networks ($k = 2, 3$). It is worth noting that with 4096 nodes a binary hypercube has a latency three times higher than the corresponding three dimensional cube with same number of nodes.

[6] Often called *base network latency*.

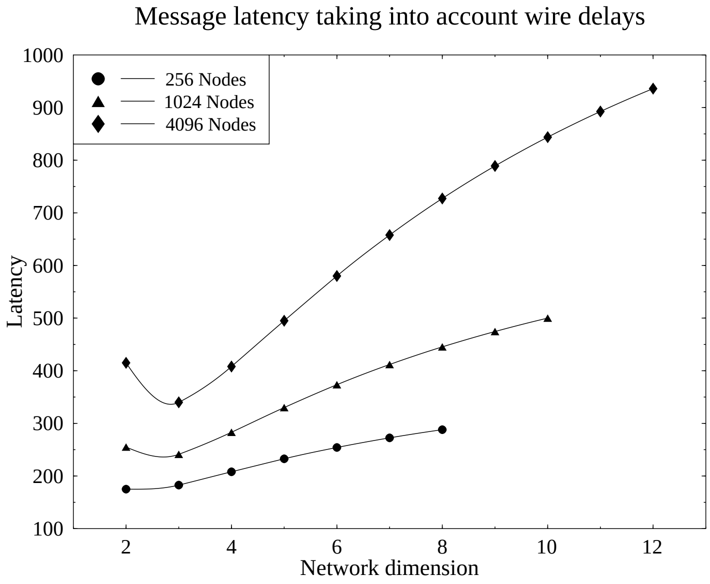


Figure 2.20: Latency for machines with 256, 1024, and 4096 nodes, varying the network dimension. The switch delay s is set to 4.0, the message length L and the channel width W are set to 160 and 8 bits.

Embedding a logical topology into the physical world poses the problem of wire density. An accepted measure of wire density is the bisection width. The bisection width for a torus connected k-ary n-cube is $2WN/k$, W being the channel width: for example in a linear array is $2W$, $2Wk$ in a mesh, and WN in a hypercube. Dally [142] studied the behavior of the base network latency keeping the bisection width fixed, normalizing its value to N, the bisection width of the binary hypercube with serial channels. In this case T_b becomes

$$T_b = (s + k^{n/2-1}) \left(n\frac{k-1}{2} + \frac{2L}{k} \right),$$

and the resulting plot is shown in figure 2.21.

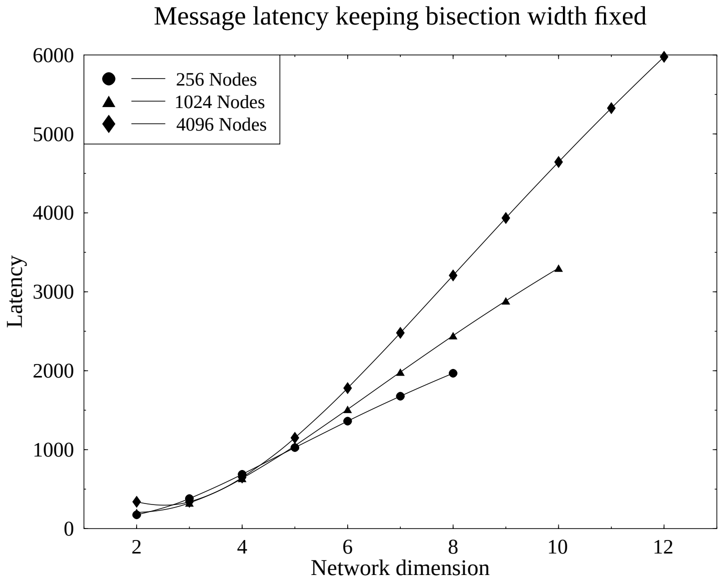


Figure 2.21: Latency for machines with 256, 1024, and 4096 nodes, varying the network dimension and keeping the bisection width fixed to N. The switch delay s is set to 4.0, the message length L to 160 bits.

The fixed bisection width constraint is often considered too strict, because indicates a cost factor more than a physical limitation [10]. The plot shown in figure 2.21 underline this penalty for the high dimensional networks.

A more realistic bound is provided by the pin count on each chip. In this case we can keep fixed the number of pins to 128 signal wires per node[7], and the base network latency can be approximated as

$$T_b = (s + k^{n/2-1}) \left(n \frac{k-1}{2} + \frac{Ln}{64} \right),$$

[7] Not counting handshake and ground lines.

with the corresponding plot in figure 2.22.

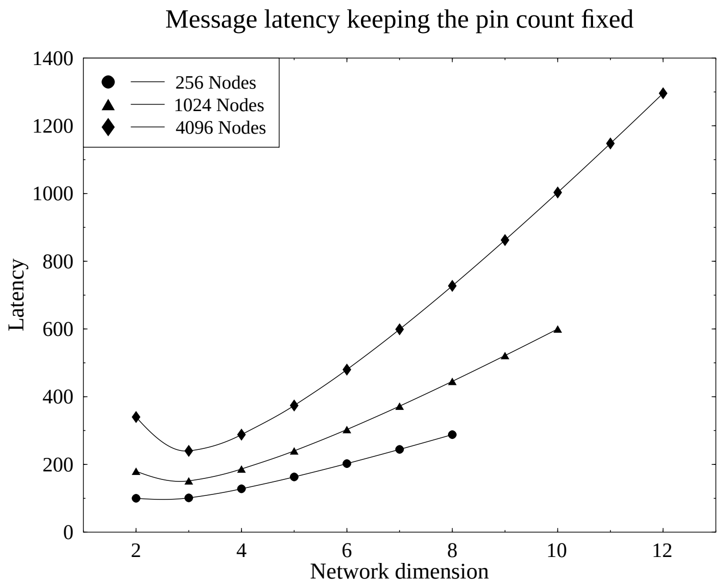


Figure 2.22: Latency for machines with 256, 1024, and 4096 nodes, varying the network dimension and keeping the pin count fixed to 128. The switch delay s is set to 4.0, the message length L to 160 bits.

All these examples show that low dimensional topologies ($k = 2, 3, 4$), outperforms high dimensional ones: they provide better base network latency, and better support for message locality, due to their wider paths. For these reasons all emerging architectures tend to use low dimensional cubes or dynamic networks built out of low dimensional switches [336] [138]. This represent a turning point with respect to the early experiences of the last decade [237].

2.5.2 Performance under uniform traffic

The analytical modeling of an interconnection networks suffers from several problems as outlined above. A simplified, but still interesting analytical model for the class of k-ary n-cubes has been proposed by Agarwal [10], which is well suited for low dimensional ones. The average latency T_c, under uniformly distributed random traffic, can be approximated as

$$T_c = \left[1 + \frac{\rho B(k_d - 1)}{(1-\rho)k_d^2}\left(1 + \frac{1}{n}\right)\right] n k_d + B,$$

where

- ρ is the average channel utilization,
- B is the packet size in flits,
- k_d is the average distance a message must travel in a network with unidirectional channels and toroidal connections ($k_d = (k-1)/2$),

When $k_d \gg 1$ (which is the case for low dimensional cubes) the above formula can be simplified to

$$T_c = n k_d + B + \frac{\rho B(n+1)}{(1-\rho)}.$$

This model is used in figure 2.23, which illustrates how packet size affects network congestion. A better performance is achieved with smaller packets. That is, smaller packets utilize the network closer to its theoretical bandwidth, without significant degradation in latency. On the other hand, for a given channel utilization smaller messages result in lower latency. Also, there is another interesting consequence: the average latency becomes less sensitive to congestion[8]. Figure 2.23 show that with 4-flit messages and a channel utilization $\rho = 0.75$ the average latency is 70 time units, about twice the latency we can get with $\rho = 0.15$. In this range the latency has a nearly linear behavior, with a small slope. Many architectures tend to use these features to have a better performance [297] [138]. The model has been recently validated by experimental evaluation on the MIT J-machine [410].

[8] Obviously, there is a practical lower bound in the size choice, due to the overhead of routing information on each packet. Packet size can not be shrunk at will if real information must be sent.

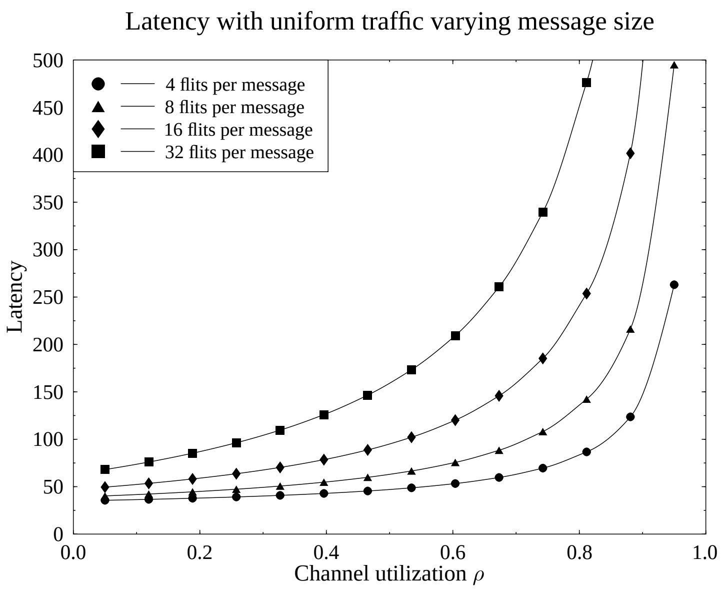


Figure 2.23: Message latency varying the channel utilization ρ, for several packet sizes in flits. The graph refers to a 32-ary 2-cube with dimension order routing, under uniform traffic.

2.6 Conclusions

The interconnection structure is a critical component in massively parallel architectures, because many algorithms are communication limited. There are many interesting solutions to this problem, with different degrees of performance, scalability and cost. Direct networks are very popular: they avoid the single bus bottleneck, providing good scalability and performance at a reasonable cost. They are cost competitive with dynamic networks as crossbar switches and multistage networks. Also, they allow the exploitation of message locality.

Though several approaches are present in the literature for each dimension involved in the definition of an interconnection network (as network type, routing and flow control), many aspects are converging towards specific solutions.

As an example, wormhole flow control proved superior in the development of routing switches. The same holds true for adaptive routing algorithms, where the Duato's algorithm offer performance close to the network capacity.

Dally and Agarwal pointed out the importance of low dimensional components for message routing, both for static and dynamic networks. Also, small sized messages, make network latency less sensitive to congestion.

All these aspects pave the way to the definition of an abstract machine [141] of massively parallel architectures: this is a key point to separate program development from interconnection networks and to guarantee portability across several different architectures.