(CS838: Topics in parallel computing, CS1221, Tue+Thu, Feb 2+4, 1999, 8:00-9:15 a.m., Pavel Tvrdik)

Section #5: Direct interconnection networks I+II

A ***direct interconnection network*** (IN) of a multiprocessor system is represented by a ***connected graph*** whose vertices represent ***processing nodes*** and edges represent ***communication links***. A processing node (PN) usually consists of one or more processors, local memory, and communication router. This section is devoted to the description and analysis of topologies and properties of important INs.

5.1 Basic notions and terminology

5.1.1 Alphabets and strings

d-ary alphabet is denoted by $\mathcal{Z}_d = \{0, 1, .., d-1\}$. The operation + modulo n is denoted by $\oplus_n$.

n-letter d-ary strings: $\mathcal{Z}_d^n = \{x_{n-1} \dots x_0; x_i \in \mathcal{Z}_d\}$, $n \geq 1$. The length of string x is denoted by $\operatorname{len}(x)$. The ***empty*** string (i.e., of length 0) is denoted by ε. The ***i-fold concatenation*** of string x is denoted by x^i.

Binary alphabet is denoted by $\mathcal{B}$. The ***inversion*** of bit b_i is $\overline{b_i} = 1 - b_1$. If $b = b_{n-1} .. b_{i+1} b_i b_{i-1} .. b_0 \in \mathcal{B}^n$, then $\operatorname{neg}_i(b) = b_{n-1} .. b_{i+1} \overline{b_i} b_{i-1} .. b_0$.

5.1.2 Graph theory

Vertex and edge set of graph G is denoted by $V(G)$ and $E(G)$, respectively. Two ***adjacent*** vertices u and v form edge $\langle u, v\rangle$. They are ***incident*** with the edge. Edges $\langle u, v\rangle$ and $\langle v, w\rangle$ are ***adjacent***.

H = ***subgraph*** of G, $H \subset G$, if $V(H) \subset V(G)$ and $E(H) \subset E(G)$.

H = ***induced subgraph*** of G if it is a ***maximal*** subgraph of G with vertices $V(H)$.

H = ***spanning subgraph*** of G if $V(H) = V(G)$.

Union of two disjoint graphs G_1 and G_2, $G_1 \cup G_2$, is a graph with vertices $V(G_1) \cup V(G_2)$ and edges $E(G_1) \cup E(G_2)$.

Cartesian product of G_1 and G_2 is graph $G = G_1 \times G_2$ with $V(G) = \{(x, y); x \in V(G_1), y \in V(G_2)\}$ and $E(G) = \{\langle (x_1, y), (x_2, y)\rangle; \langle x_1, x_2\rangle \in E(G_1)\} \cup \{\langle (x, y_1), (x, y_2)\rangle; \langle y_1, y_2\rangle \in E(G_2)\}$. See an example bellow.

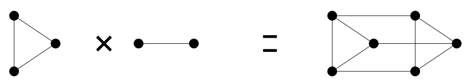


Figure 5.1. An example of a cartesian product

Degree of vertex u, $\deg_G(u)$, is the number of neighbors of u.

Degree set of graph G, $\deg(G)$, is the set $\{\deg_G(u); u \in V(G)\}$.

Maximum degree of G is $\triangle(G) = \max(\deg(G))$.

Minimum degree of G is $\delta(G) = \min(\deg(G))$.

k-regular graph G has $\triangle(G) = \delta(G) = k$.

Connected graph: Every pair u, v of vertices is joined by a ***path*** $P(u, v)$, a sequence of adjacent edges.

Path length $\operatorname{len}(P(u, v))$, is the number of edges $P(u, v)$.

Distance between vertices u and v, $\operatorname{dist}_G(u, v)$, is the length of a shortest path joining u and v.

Average distance in G, $\operatorname{dist}_G(u, v)$, is $\Sigma_{u,v} \operatorname{dist}_G(u, v)/(N(N-1))$.

Diameter of G, $\phi(G)$, is the maximum distance between any two vertices of G.

Cycle is a closed path.

Vertex-disjoint paths $P_1(u,v)$ and $P_2(u,v)$ have no vertices in common except for u and v

Edge-disjoint paths $P_1(u,v)$ and $P_2(u,v)$ have no edges in common.

Connectivity*: *(Vertex) connectivity of G, $\kappa(G)$, is the ***minimum*** number of vertices whose removal results in a disconnected graph. ***Edge connectivity*** of G, $\lambda(G)$, is the ***minimum*** number of edges whose removal results in a disconnected graph. It follows that

$$\kappa(G) \leq \lambda(G) \leq \delta(G).$$

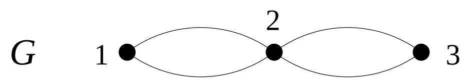


Figure 5.2. Graph G with $\kappa(G) = 1$ and $\lambda(G) = \delta(G) = 2$.

k***-connected graph*** has $\kappa(G) = k$. Similarly, k***-edge-connected*** graph has $\lambda(G) = k$.

***Optimal connectivity*:** $\kappa(G) = \lambda(G) = \delta(G)$.

***Menger's theorem*:** *Between any two distinct vertices of G, there are at least $\kappa(G)$ vertex-disjoint paths and at least $\lambda(G)$ edge-disjoint paths.*

Fault diameter of a connected graph G is the maximum over the lengths of all the shortest vertex-disjoint paths between any two vertices in G. Similarly, we can define the ***fault distance*** between any two vertices.

Bipartite graph*:** There exists a ***bipartition (or 2-***coloring***) of its vertex set V, which is a partitioning of V into two disjoint subsets V_1 and V_2 such that each edge of G has one vertex in V_1 and the other vertex in V_2.

Balanced bipartite graph has a bipartition (V_1, V_2) with $|V_1| = |V_2|$.

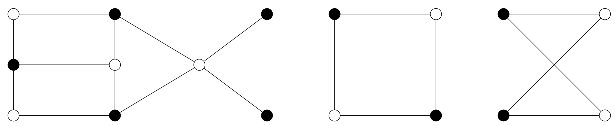


Figure 5.3. Examples of bipartite graphs, the first one is not balanced

Hamiltonian path/cycle of a connected G is a path/cycle joining all vertices of G. A graph having a hamiltonian cycle is ***hamiltonian***. *A bipartite graph can have a hamiltonian cycle only if it is balanced.*

(Edge) bisection width of G, $\mathrm{bw_e}(G)$, is the smallest number of edges removal of which divides G into two parts of equal size (up to one vertex).

Vertex bisection width of G, $\mathrm{bw_v}(G)$, is the smallest number of vertices removal of which divides G into two parts having at most $\lceil |V|/2 \rceil$ vertices each. To construct a minimal bisection is an NP-complete problem.

***Isomorphism*:** Two graphs are isomorphic if they can be made identical by relabeling vertices.

Automorphism of G is any isomorphic mapping of G to itself.

Vertex-symmetric graph looks the same independently of from which ***vertex*** you look at it, i.e., for any two distinct vertices u and v, there is an automorphism of the graph sending u to v.

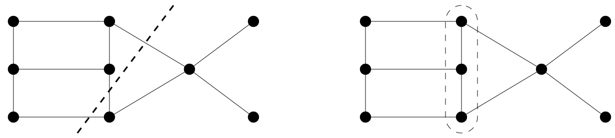


Figure 5.4. An example of a graph with both edge and vertex bisection width 3

Edge-symmetric graph looks the same independently of from which ***edge*** you look at it, i.e., for any two distinct edges e_1 and e_2 in $E(G)$, there is an automorphism of G sending e_1 to e_2.

Graph G on Figure 5.2 is edge symmetric, but it is not vertex symmetric. It becomes vertex symmetric if the vertices 1 and 3 are merged. A vertex symmetric graph must be ***regular***.

Digraphs = Oriented graphs: A digraph G has **arc set** $A(G)$. Arc $\langle u \rightarrow v \rangle$ is ***incident from*** u ***to*** v $u, v \in V(G)$. Vertex u (v) is ***incident from*** (***to***) arc $\langle u \rightarrow v \rangle$, respectively. Arcs $\langle u \rightarrow v \rangle$ and $\langle v \rightarrow w \rangle$ are ***adjacent***. The ***in-degree*** and ***out-degree*** of $u \in V(G)$, denoted by $\deg_G^{\text{in}}(u)$ and $\deg_G^{\text{out}}(u)$, is the number of arcs incident to and incident from u, respectively. The other notions are derived similarly, digraphs have oriented paths (dipaths), oriented diameters, strong connectedness, and strong connectivity.

While describing INs, we will often use terms ***nodes*** and ***links*** (or ***channels***) instead of vertices and edges, terms ***input*** (***output***) links or channels, respectively, instead of arcs incident from (incident to) a vertex. We will often use the terms vertices/nodes or edges/links interchangeably.

All logarithms are ***binary*** if not stated otherwise.

5.2 Requirements on interconnection networks

An IN should transfer a ***maximum*** number of messages in the ***shortest time*** with ***minimum cost*** and ***maximal reliability***. Clearly, any design of an IN is a tradeoff of various ***contradictory requirements***. The most important requirements are the following:

Small diameter and small average distance: Small average distance allows small communication ***latency***, especially for ***distance sensitive*** routing, such as ***store-and-forward***. But it is also crucial for ***distance insensitive*** routing, such as ***wormhole routing***, since short distances imply less used links and buffers, and therefore less communication ***contention***.

Small and fixed vertex degree: Small and constant vertex degree allows simple and low-cost universal (i.e., independent of the network size) routers which amortizes the design costs. On the other hand, it implies less links and lower connectivity and larger distances. Ideally, we would like to get ***low constant vertex degree*** and at most ***logarithmic diameter*** simultaneously. Given N-vertex graph such that $\triangle(G) \leq k$ for some constant k, the number of vertices reachable in G within i steps from any vertex is $O(k^i)$. Hence, $N = O(k^{\phi(G)})$, which is equivalent to $\phi(G) = \Omega(\log N)$. *Fixed-degree networks cannot have better than logarithmic diameter.*

Large bisection width: Many problems can be solved in parallel using ***binary divide-and-conquer***: split the input data set into two halves and solve them recursively on both halves of the IN in parallel, and then merge the results in both halves into the final result. Small bisection width implies ***low bandwidth*** between both halves and it can slowdown the final merging phase. On the other hand, a large bisection width is undesirable for a VLSI design of the IN, since it implies a lot of ***extra chip wires***.

High connectivity and fault tolerance: The network should provide ***alternative paths*** for delivering messages in case of link or node ***faults*** or ***communication congestions***. ***Large packets*** can be delivered faster if they can be ***split*** into smaller chunks sent along disjoint paths.

Small fault average distance and diameter: To have these alternative or parallel disjoint paths as short as possible, small fault average distances and small fault diameter are naturally desirable.

Hamiltoniaty: The existence of at least a hamiltonian path is not the most important requirement, but it is useful whenever we need to label processors with numbers $1, \ldots, p$ so that adjacent processors get successive numbers (for example in sorting algorithms).

Hierarchical recursivity*:** INs are usually defined using some independent parameters, called ***dimensions. The set of all graphs of different dimensions based on a given definition forms a ***topology***. A topology is ***hierarchically recursive*** if a higher-dimensional graph contains lower-dimensional graphs with the same topology as subgraphs. Many INs used in parallel systems are hierarchically recursive, topologies based on cartesian product are one example. Recursiveness makes the design and technology of manufacturing INs easier. Large scale problems that can be solved in parallel by recursive decomposition into smaller subproblems can be easily and efficiently mapped on the hierarchically recursive topologies using induction.

Incremental extendability and incremental scalability*:** If the definition of the topology allows graphs of any size, the topology is said to be ***incrementally extendable. There are incrementally extendable topologies, and some are at least partially extendable, since they allow size granularity only greater than 1. Some hierarchically recursive topologies allow graphs of specific discrete sizes, such as powers of two. If a topology is incrementally extendable, a very important question is how the structure of an instance of size n differs from the structure of an instance of size $n + k$ for some integer constant $k \geq 1$. An $(n + k)$-vertex instance can be obtained from a n-vertex one by removing $r(k)$ edges (to get a subgraph of the larger instance) and by adding additional vertices and corresponding edges to this subgraph. If $r(k) = O(k)$, the topology is said ***incrementally scalable***. Very few topologies are incrementally scalable. For example, 2-D meshes are incrementally extendable, but not scalable.

***Symmetry*:** This is a very important requirement. Many IN topologies are vertex or edge symmetric. Intuitively, a symmetric network is easy to understand and the design of parallel and communication algorithms is very much easier, since it is irrelevant where the computation and/or communication starts or in which directions it will evolve. Also, the symmetry is helpful for solving issues related to VLSI design.

Support for routing and collective communication*:** This is a crucially important property and we will devote it several lectures. The network topology should enable an simple shortest path routing, so that the basic routing algorithm could be implemented in hardware. Equally important parameters of an IN are complexities of ***permutation routing and ***one-to-all*** and ***all-to-all*** communication operations. The design of efficient communication algorithms is very simplified if the topology is symmetric and it also depends on the communication technology and router architecture.

Embeddability of and into other topologies*:** The efficiency of a parallel algorithm running on a parallel machine depends on the similarity between the process graph and the underlying physical IN topology. We need a suitable mapping or ***embedding of the process graph into the topology. Desirable are those topologies which are able to simulate efficiently other topologies.

***Simple VLSI or 3-D layout*:** Any VLSI implementation of an IN implies VLSI-related requirements, such as easy mapping to rectangular grid, decomposition of a large IN into building blocks so that the lengths and numbers of inter-chip wires is minimal (recall the bisection width above). Except for the bisection width, the discussion of VLSI design issues is beyond the scope of this course.

In the rest of the section, we give a survey of definitions and properties of several well-known topologies for INs. Some topologies are understood better than some others so that the description of properties will vary for various topologies.

5.3 Mesh-based topologies

This family of topologies, also called ***strictly orthogonal***, include hypercubes, meshes, and tori. These are the most important interconnection topologies due to their simplicity and they are also the best understood topologies.

5.3.1 Binary hypercube of dimension n, Q_n

The n-***dimensional binary hypercube*** (or n-***cube***) is a graph with 2^n vertices labeled by n-bit binary strings, with edges joining two vertices whenever their labels differ in a single bit (see Figure 5.5).

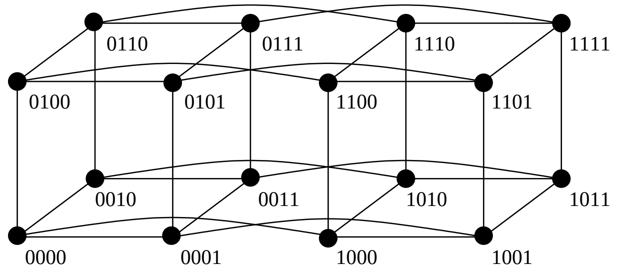


Figure 5.5. Hypercube Q_4

$V(Q_n) = \mathcal{B}^n$	$\lvert V(Q_n)\rvert = 2^n$
$E(Q_n) = \{\langle x, \mathrm{neg}_i(x)\rangle; x \in V(Q_n), 0 \le i < n\}$	$\lvert E(Q_n)\rvert = n2^{n-1}$
$\phi(Q_n) = n$	$\deg(Q_n) = \{n\}$
$\mathrm{bw_e}(Q_n) = 2^{n-1}$	

- The hypercube is a *regular balanced bipartite hamiltonian* graph.
- The distance between two vertices u and v in Q_n is the *Hamming distance.*
- It has *logarithmic* diameter. The average distance in Q_n is $n/2$.
- It is *not* a constant-degree topology, the vertex degree grows logarithmically with the number of vertices. If the number of nodes of a parallel computer doubles, the degree of each vertex increases by one. A router with k communication ports can be used for building a machine with at most 2^k nodes. This problem is solved by constant-degree and logarithmic-diameter derivatives of the hypercube, called ***hypercubic networks***, see Page 9.
- The hypercube has the *largest possible bisection width.* Each vertex in one half is adjacent to its ***image*** in the other half.
- The connectivity of Q_n is n, which is *optimal.* If $k \le n$ is the distance between vertices u and v, then there are n vertex-disjoint (therefore edge-disjoint) paths between u and v such that k of these paths have length k and $n-k$ paths have length $k+2$.
- The *fault diameter* of Q_n is n. If we remove $n-1$ edges from Q_n, there still exists a path of length at most n between any two vertices.
- Fault tolerance is also excellent, there are $k!$ different shortest paths between two vertices in distance k, since $k!$ is the number of permutations of k coordinates in which the addresses differ.
- The hypercube is *hierarchically recursive.* Q_n is a cartesian product of ***subcubes***:

$$Q_n = Q_k \times Q_{n-k}, \qquad k \in \{1, \dots, n-1\}.$$

  There are $\binom{n}{k}$ ways how to make this decomposition. In particular, for any $j \in \mathcal{N}$ there exists a ***canonical decomposition*** of Q_n into two $(n-1)$-cubes $Q_{n-1}[j=0]$ and $Q_{n-1}[j=1]$ (This gives the optimal bisection width.) For example, since $Q_6 = Q_3 \times Q_3$, we can view a Q_6 as a 3-cube, whose vertices are 3-cubes (Figure 5.6 shows a simplified picture of that). But also $Q_6 = Q_2 \times Q_4 = Q_2 \times Q_2 \times Q_2$, and so on. Each k-dimensional subcube S of Q_n is uniquely specified by its ***address*** $s_{n-1} \dots s_1 s_0$, where $s_i \in \{0, 1, *\}$, with exactly k appearances of the ***don't care*** symbol $*$. 0-dimensional subcubes are vertices and subcubes are generalized "internally structured" vertices. Vertices correspond to minterms and subcubes to terms in Boolean algebra.
- The hypercube topology is *not* incrementally extendable, it is defined only for powers of two. However, many incrementally extendable generalizations have been proposed, such as ***incomplete hypercubes***.
- The hypercube is both *vertex-* and *edge-symmetric graph.* The automorphism group of Q_n consists of $2^n \times n!$ translations and rotations. A translation is a systematic renaming of vertices. Let XOR denote the bitwise XOR operation. ***Translation*** from u to v is a mapping $\forall x \in V(Q_n)\ (x \mapsto (u \,\mathrm{XOR}\, v))$. A ***rotation*** is a permutation (renaming) of dimensions.

- The basic routing algorithm is *trivial* and *optimal*, usually hardwired, and called ***e-cube*** routing: The bits in which the sender and receiver differ are compared in a fixed order, usually left to right.
- There exist *optimal* algorithms for collective communication in almost all communication models.
- The hypercube can also *simulate efficiently almost any other topology.*

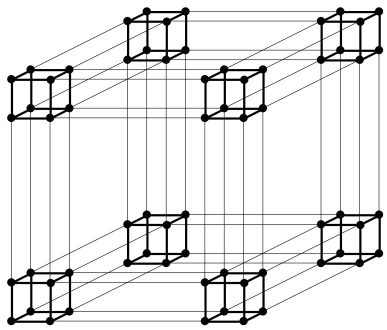


Figure 5.6. $Q_6 = Q_3 \times Q_3$.

The notion of subcubes is extremely important for using a hypercube multiprocessor as a multiuser multitasking machine. Users declare the numbers of required nodes for their tasks and the operating system allocates, if possible, subcubes of the appropriate size, which are released upon completion of the tasks. This is called the ***subcube allocation problem*** and there exist many algorithms for it which must maintain lists of allocated and free subcubes. Dynamic allocation and releasing of subcubes lead to the ***fragmentation*** problem in the hypercube similar to segment fragmentation of dynamic or virtual memory.

The hypercube is the best topology for implementing most parallel and communication algorithms. It serves as a testbed for parallel feasibility of problems on distributed memory architectures. Its only drawback is the logarithmic vertex degree and lack of scalability.

Several families of older massively parallel multiprocessors were based on the hypercube: nCUBE series 1 and 2, Intel's iPSC/2 and iPSC/860, TMC's CM-2, and several others. Currently, SGI Origin's interconnection network uses hypercube topology.

5.3.2 n-dimensional mesh of dimensions $z_1, z_2, .., z_n$, $M(z_1, z_2, .., z_n)$

Given integers $z_i \geq 2$, $1 \leq i \leq n$, the n-dimensional mesh with side lengths z_i, $M(z_1, z_2, .., z_n)$, is defined as follows (Figure 5.7(a) shows mesh $M(3, 3, 4)$).

$$V(M(\ldots)) = \{(a_1, a_2, .., a_n); 0 \leq a_i \leq z_i - 1 \,\forall i \in \{1, .., n\}\} \quad |V(M(\ldots))| = \prod_{i=1}^{n} z_i$$
$$E(M(\ldots)) = \{\langle(.., a_i, ..), (.., a_i + 1, ..)\rangle; 0 \leq a_i \leq z_i - 2\} \quad |E(M(\ldots))| = \sum_{i=1}^{n}(z_i - 1)\prod_{\substack{j=1 \\ j \neq i}}^{n} z_j$$
$$\phi(M(\ldots)) = \sum_{i=1}^{n}(z_i - 1) = \Omega(\sqrt[n]{|V(M(\ldots))|})$$
$$\deg(M(\ldots)) = \{n, .., n + j\},\ j = |\{z_i; z_i > 2\}|$$
$$\mathrm{bw_e}(M(\ldots)) = \begin{cases} (\prod_{i=1}^{n} z_i)/\max_i z_i & \text{if } \max_i z_i \text{ is even,} \\ \Omega((\prod_{i=1}^{n} z_i)/\max_i z_i) & \text{otherwise.} \end{cases}$$

- When dealing with a mesh, we usually assume that its dimension n is *fixed*. If we want to change its size, we change the side lengths. All the properties explained in what follows are stated under the assumption of fixed mesh dimension.
- The most practical meshes are, of course, *2-D* and *3-D* ones: they have trivial VLSI or 3-D design (2-D meshes are planar) and allow simple and natural mapping of 2-D and 3-D space problems with locality of interprocessor interactions (computer graphics and virtual reality, fluid mechanics, modeling of 3-D systems in earth sciences or astronomy, etc). They are also natural, even though not optimal, topologies for linear algebra problems (parallel solvers of linear or differential equations). 1-D meshes are called ***linear arrays*** and they are incrementally scalable.
- The mesh $M(k, k, .., k)$ is called ***k-ary n-cube***.

- A mesh is a *bipartite* (not necessarily balanced) graph. If at least one side has even length, the mesh has a *hamiltonian cycle*. A hamiltonian path exists always.
- Meshes are *not regular*, but the degree of any vertex is *bounded* by $2n$. Of course, the degree of a corner vertex is less than the degree of an internal vertex.
- Therefore, meshes are *not vertex symmetric*. This complicates the design of some parallel and communication algorithms, but on the other hand, there are problems taking advantage of the asymmetry between boundary and internal vertices, e.g., parallel finite difference schemes for solving differential equations in 2-D and 3-D space.
- The distance between two vertices is the sum of differences in corresponding coordinates.
- The diameter is *n-th root* of the size of graph. It is large for lower-dimensional meshes, and therefore, they are not optimal for algorithms with global interactions, e.g., parallel sorting, or problems where global broadcasting or permutations are required, such as in parallel solvers of linear equations. So, 2-D or 3-D mesh are simple extendable INs with small fixed vertex degree.
- The exact value of bisection width is known only if $\max_i z_i$ is even. Then removing the middle edges of linear paths in the longest-side direction results in a bisection of minimal size. However, if $\max_i z_i$ is odd, the exact value of the bisection width is not known for $n \geq 4$. However, it is easy to calculate it for 2- or 3-D meshes. For example, $\mathrm{bw_e}(M(11,9,6)) = 64$. (Can you show that?) Lower-dimensional meshes have very low bisection width, which creates a bottleneck for many parallel mesh-based algorithms.
- The connectivity is *optimal*. The proof is similar to that for hypercubes. Also, the shortest path redundancy is optimal.
- The fault diameter is *optimal*, equal to the nonfaulty diameter. Fault distances are by at most 4 larger than nonfaulty ones.
- Meshes are also *hierarchically recursive*. They can be decomposed into cartesian products of lower-dimensional meshes, up to the ***canonical decomposition*** into a cartesian product of linear arrays: $M(z_1, z_2, .., z_n) = M(z_1, z_2, .., z_{n-1}) \times M(z_n) = M(z_1, .., z_i) \times M(z_{i+1}, .., z_j) \times M(z_{j+1}, .., z_n) = M(z_1) \times M(z_2) \times \cdots \times M(z_n)$. Submeshes are also denoted by strings with don't-care symbols.
- Meshes are *incrementally extendable*, but *not* incrementally scalable. You can define a mesh of any size, but the amount of edges that must be removed from a smaller one to construct a larger one, can be very large.
- The basic shortest-path routing in general meshes is also trivial. It is called ***dimension ordered*** routing. Again, addresses of the sender and receiver are inspected in a prespecified order and the message is sent via the first port corresponding to a coordinate in which the addresses differ. In 2- and 3-D meshes, this is called XY and XYZ ***routing***.
- Many optimal communication and parallel algorithms have been developed for meshes (optimal in the sense of matching the mesh lower bounds due to large diameter, low bisection width, etc), namely for 2-D and 3-D meshes. Many of them will be treated in detail in this course.
- Besides standard meshes, various modifications have been proposed, such reconfigurable meshes, meshes of buses, and so on. We will not deal with these variants here.

The most important mesh-based parallel computers are Intel's Paragon (2-D mesh) and MIT J-Machine (3-D mesh). Also transputers used 2-D mesh interconnection. Processors in mesh-based machines are allocated by submeshes and the *submesh allocation* strategy must handle possible *dynamic fragmentation* and *compaction* of the global mesh network, similarly to hypercube machines.

5.3.3 n-dimensional torus of dimensions $z_1, z_2, .., z_n$, $T(z_1, z_2, .., z_n)$

Given integers $z_i \geq 2$, $1 \leq i \leq n$, the n-dimensional torus $T(z_1, z_2, .., z_n)$, also called ***toroidal*** or ***wrapped mesh***, with side lengths $z_1, z_2, .., z_n$, is defined as follows.

$$\begin{array}{ll}
V(T(\ldots)) = V(M(\ldots)) & |V(T(\ldots))| = \prod_{i=1}^{n} z_i \\
E(T(\ldots)) = \{\langle(.., a_i, ..), (.., a_i \oplus_{z_i} 1, ..)\rangle; 0 \leq a_i < z_i\} & |E(T(\ldots))| = n \times \prod_{i=1}^{n} z_i \\
\phi(T(\ldots)) = \sum_{i=1}^{n} \left\lfloor \frac{z_i}{2} \right\rfloor & \deg(T(\ldots)) = \{2n\} \\
\mathrm{bw_e}(T(\ldots)) = 2\,\mathrm{bw_e}(M(\ldots)) &
\end{array}$$

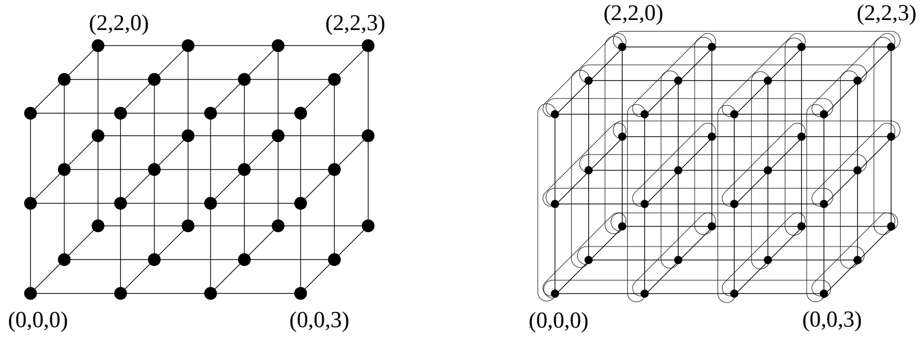


Figure 5.7. Mesh $M(3,3,4)$ and torus $T(3,3,4)$.

Figure 5.7(b) shows a 3-dimensional torus $T(3,3,4)$. Adding wrap-around edges changes dramatically many structural properties, even though many feature remain similar to meshes. We will highlight some of them.

- A 1-dimensional torus is simply a *cycle* or *ring*.
- The torus $T(k,k,..,k)$ is called ***k*-ary *n*-torus**.
- Tori are *bipartite* iff all side lengths are even. But then, they are *balanced*.
- Any torus has a *hamiltonian cycle*.
- Tori are *regular* and *vertex symmetric*. The automorphism group consists of translations. k-ary n-tori are also *edge symmetric* and have rotations as automorphisms.
- The diameter and average distance of a torus drops to nearly *one half* compared to a mesh with the same dimensionality, and the connectivity and the bisection width *doubles*. Hence, the connectivity of n-dimensional torus, $2n$, is also *optimal*.
- They are also *hierarchically recursive*, again thanks to cartesian product structure. Each torus can be canonically decomposed into a cartesian product of cycles.
- Scalability properties are even slightly worse here than at meshes.
- Cyclic structure of tori makes shortest path routing algorithms more complicated, but the basic approach remains *dimension ordered* routing.
- Similarly to meshes, many fundamental problems are known to have topologically optimal solutions on tori and we will demonstrate many of them here.

5.3.4 Comparisons of hypercubes, meshes, and tori

- $M(2,2,..,2)$ is isomorphic Q_n
- n-dimensional meshes and tori are generalizations of Q_n
- For some k and n, $k < n$, a k-dimensional mesh/torus can be a subgraph of Q_n.

For example, Figure 5.8 illustrates that torus $T(4,4)$ is isomorphic to Q_4. Table 5.3.4 shows the tradeoff

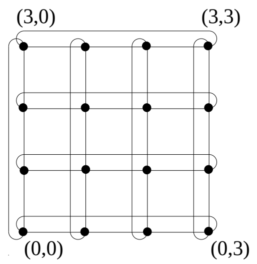


Figure 5.8. $T(4,4) = Q_4$.

	$M(8,8,4)$	$T(8,8,4)$	Q_8
$\phi()$	17	10	8
$\|E()\|$	640	768	1024
$\mathrm{bw_e}()$	32	64	128

Table 5.1. Comparison of several characteristics of meshes, tori, and hypercubes.

between the diameter, number of edges, and bisection width of $M(8,8,4)$, $T(8,8,4)$, and Q_8. All these graphs have the same size $|V()| = 256$ and $M(8,8,4) \subset T(8,8,4) \subset Q_8$.

The tori are likely to become widely used interconnection networks. For example, Cray/SGI T3D and T3E and Convex Exemplar are based on 3-D torus, Intel/CMU iWarp uses 2-D torus, and several commercial machines use rings as a primary interconnect.

5.4 Hypercubic topologies

Hypercubic networks are logarithmic-diameter and constant-degree derivatives of the hypercube. The price to be paid for the constant vertex degree is worse extendability: the size of instances becomes $n2^n$ or similar. Therefore, they grow even faster with increasing dimension than the hypercube. They share many properties, such as bisection width $\Omega(N/\log N)$. Let us shortly describe two representants: ***cube-connected cycles*** and ***butterflies***.

5.4.1 Cube-connected cycles of dimension n, CCC_n

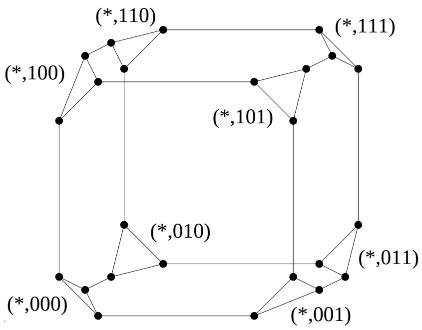


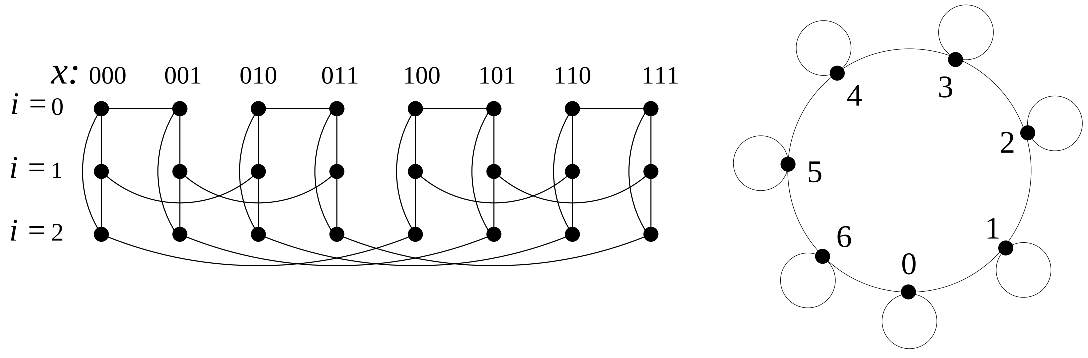
Figure 5.9. Cube Connected Cycles CCC_3.

$V(CCC_n) = \{(i,x); 0 \le i < n \wedge x \in \mathcal{B}^n\}$ $\qquad |V(CCC_n)| = n2^n$
$E(CCC_n) = \{\langle (i,x),(i \oplus_n 1, x)\rangle, \langle (i,x),(i,\mathrm{neg}_i(x))\rangle; (i,x) \in V(CCC_n)\}$ $\qquad |E(CCC_n)| = n2^{n-1} + n2^n$
$\phi(CCC_n) = (2n-2) + \lfloor n/2 \rfloor$ for $n > 3$, $\phi(CCC_3) = 6$ $\qquad \deg(CCC_n) = \{3\}$
$\mathrm{bw_e}(CCC_n) = 2^{n-1}$

CCC_n is constructed from Q_n by replacing each hypercube vertex with a cycle of n vertices so that each vertex takes care of just one hypercube direction. See Figure 5.9 where $(*,x)$ stands for $\{(0,x),(1,x),(2,x)\}$. Figure 5.10 shows other possible ways how to view CCC topology. Part (b) is a useful simplification, CCC is viewed as a pancake made of all cycles, all the cycles look alike, and all hypercube edges are projected into vertex loops.

- CCC_n is *balanced bipartite* iff n is even.
- Any CCC_n has a *hamiltonian cycle*.
- CCC is a *regular* and *vertex-symmetric* topology. Figure 5.10(b) shows it clearly. It is *not* an edge-symmetric graph, there are two kinds of edges: ***hypercube*** edges and ***cycle*** edges.

- It has *optimal* connectivity 3. Between two arbitrary vertices, there are at most two shortest paths.
- Shortest path routing is based on e-cube routing and Figure 5.10(b) is helpful in understanding it.
- In contrast to the hypercube, CCC is not hierarchically recursive. We can split a CCC_n into two halves which are very similar to CCC_{n-1}, but they have one abundant vertex in each cycle.



(a) CCC_3 (b) CCC_7

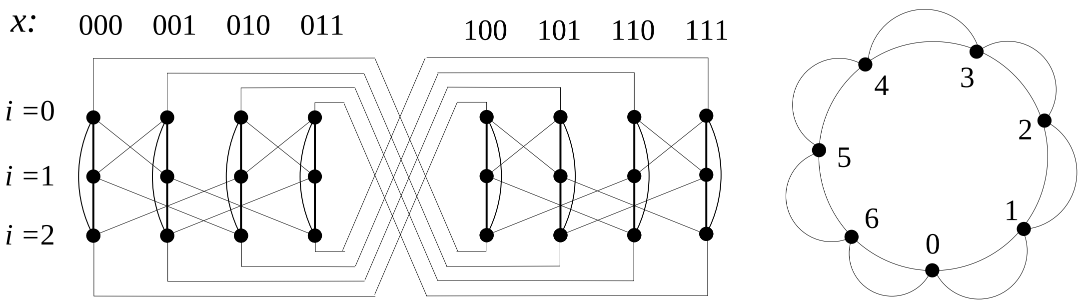
Figure 5.10. Another view of CCC.

5.4.2 Butterfly of dimension n

Actually, there are two kinds of butterfly networks, wrapped and ordinary. The ***wrapped butterfly***, wBF_n, is defined as follows:

$$V(wBF_n) = \{(i,x); 0 \leq i < n \wedge x \in \mathcal{B}^n\} \qquad |V(wBF_n)| = n2^n$$
$$E(wBF_n) = \{\langle(i,x),(i \oplus_n 1, x)\rangle, \langle(i,x),(i \oplus_n 1, \mathrm{neg}_i(x))\rangle \mid (i,x) \in V(wBF_n)\} \qquad |E(wBF_n)| = n2^{n+1}$$
$$\phi(wBF_n) = n + \lfloor \tfrac{n}{2} \rfloor \qquad \deg(wBF_n) = \{4\}$$
$$\mathrm{bw_e}(wBF_n) = 2^n$$

If each cycle of wBF_n shrinks to a single vertex and redundant edges are removed, we get Q_n (see Figure 5.11). Wrapped butterfly topology has basically the same properties as CCC, except that it has more edges, and therefore larger bisection width and lower diameter.



(a) wBF_3 (b) wBF_7

Figure 5.11. Wrapped butterflies

An n-dimensional ***ordinary butterfly***, oBF_n, can be made form wBF_n by replacing n-vertex cycles with $(n+1)$-vertex paths (or in other words, an ordinary butterfly is a hypercube with each vertex unfolded into a path).

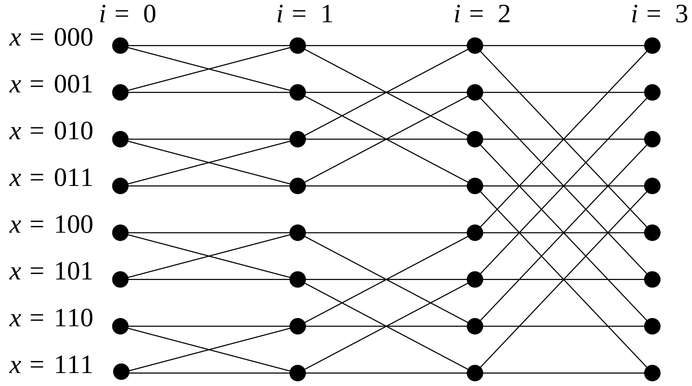


Figure 5.12. Ordinary butterfly oBF_3.

$V(oBF_n) = \{(i,x); 0 \le i \le n \wedge x \in \mathcal{B}^n\}$ $\quad |V(oBF_n)| = (n+1)2^n$
$E(oBF_n) = \{\langle (i,x),(i+1,x)\rangle, \langle (i,x),(i+1,\mathrm{neg}_i(x))\rangle \mid (i,x) \in V(oBF_n), i < n\}$ $\quad |E(oBF_n)| = n2^{n+1}$
$\phi(oBF_n) = 2n$ $\quad \deg(oBF_n) = \{2,4\}$
$\mathrm{bw_e}(oBF_n) = 2^n$

- The vertices of oBF_n are organized into ***columns*** (***stages***) i, $0 \le i \le n$, and ***rows*** x, $0 \le x \le 2^n - 1$. There are also two kinds of edges in the ordinary butterfly: ***straight*** edges and ***cross*** (***hypercube***) edges (see Figure 5.12). Edges linking vertices of stages i and $i+1$ are said to be ***of stage*** i.
- The ordinary butterfly is *not* vertex symmetric, it is *not* even regular.
- It is not *hamiltonian*.
- On the other hand, it is *hierarchically recursive*: oBF_n contains two oBF_{n-1} as subgraphs.
- The connectivity is 2, which is *optimal*.
- There exists only one shortest path between any pair of vertices $(0,x)$ and (n,y). The path traverses each stage exactly once using a cross edge of stage i iff the row numbers x and y differ in the i-th bit. It follows that the basic shortest-path routing is basically the e-cube routing from hypercubes, see Page 6.

5.5 Tree-based topologies

Process graphs have often tree-like structure. However, the bottleneck at the root and poor connectivity and robustness disqualifies any simple complete tree as a potential general purpose IN topology. Fortunately, tree-like graphs can be efficiently mapped on most usual INs. The only commercially important tree-based parallel machine have been TMC's CM-5 with its famous ***fat tree*** and some special-purpose ***pyramidal machines***. In this course, we will describe an interesting hybrid of meshes and trees.

5.5.1 2-D mesh of trees of height n, MT_n

2-D and 3-D meshes suffer from *large diameter*, whereas trees suffer from *small connectivity* and *bisection width*. 2-D mesh of trees is a hybrid topology based on the 2-D mesh and complete binary tree, which has the diameter of the tree topology (i.e., logarithmic) and the bisection width of the 2-D mesh topology (i.e., square root of the size). For integer $n \ge 1$, let $N = 2^n$. Then MT_n is defined as follows:

$V(MT_n) = (\mathcal{B}^n \times \cup_{i=0}^n \mathcal{B}^i) \cup (\cup_{i=0}^n \mathcal{B}^i \times \mathcal{B}^n)$ $\quad |V(MT_n)| = 3N^2 - 2N$
$E(MT_n) = \{\langle (x,y),(xa,y)\rangle; \mathrm{len}(x) < n \wedge a \in \mathcal{B}\}$ $\quad |E(MT_n)| = 4N(N-1)$
$\quad \cup \{\langle (x,y),(x,ya)\rangle; \mathrm{len}(y) < n \wedge a \in \mathcal{B}\}$ $\quad \mathrm{bw_e}(MT_n) = N = \Theta(\sqrt{|V(MT_n)|})$
$\deg(MT_n) = \{2,3\}$ $\quad \phi(MT_n) = 4\log N = 4n$

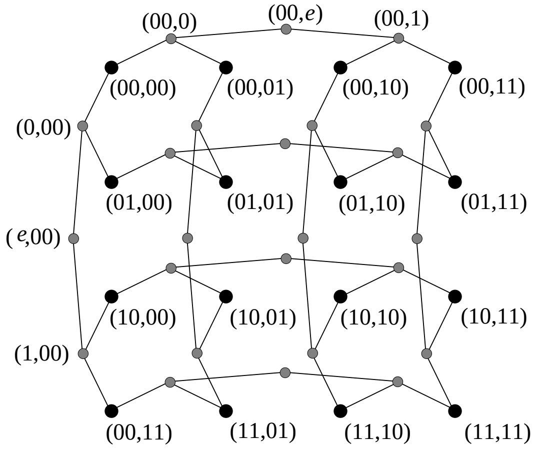


Figure 5.13. Mesh of trees MT_2.

- The mesh-of-trees topology can be defined for any mesh $M(2^k, 2^l)$. It can be even defined for any n-dimensional mesh with side lengths equal to powers of two. We will discuss only the 2-D square case for simplicity.
- Informally, a mesh-of-trees is a 2-D mesh whose linear rows and columns are replaced with ***row*** and ***column*** complete binary ***trees*** of height n, CBT_n. The original mesh vertices are *leaves* in the intersections of the row and column trees and the additional vertices are exactly the internal vertices of the trees. It follows that $MT_n \subset CBT_n \times CBT_n$.
- An N-vertex mesh-of-trees is constant vertex degree $O(\log N)$-diameter and $\Omega(\sqrt{N})$-bisection width topology.
- The mesh-of-trees topology is *bipartite*, but *not* balanced, therefore it is *not* hamiltonian.
- It is *hierarchically recursive*. By removing row- or column-tree roots, it decomposes into disjoint submeshes-of-trees. Low-level submeshes-of-trees can be used for local computations and the root nodes can be used for global communication and coordination.
- It is *neither* regular *neither* vertex symmetric.
- Its extendability is rather poor.
- Shortest-path routing is simple, it alternates routing in 2-D meshes and trees. Assume source vertex u and destination vertex v.
  - If u is in a *row* tree and v in a *column* tree, the shortest path from u to v is uniquely defined via the intersection leaf of both trees.
  - If u and v are in two *different row* trees T_u and T_v, compare the level of u and v in their trees. If u is lower than v, then go down in T_u to any leaf in the subtree, use the corresponding column tree to get to T_v, and use the tree routing in T_v to get to v (hence pass through the root of the smallest common subtree of v and of the homothetic image of u in T_v). If u is higher than v, then go first to the image of v in T_u and via any leaf get to T_v then. The number of different shortest paths depends on the heights of the vertices.
  - The routing between two internal nodes of column trees is similar.

No commercial machine has been built upon this topology yet.

5.6 Shuffle-based topologies

The adjacency in orthogonal and hypercubic topologies is based on incrementing and/or inverting coordinates. In shuffle topologies, the adjacency is based on ***shifting*** the whole strings, typical operation is left shift and left rotation by one position, which enforces orientation on edges. Since most of notions and results apply equally to both directed and undirected versions of these topologies and digraphs are regular in contrast to undirected graphs, we will concentrate on digraphs only.

Shuffle-based topologies have very rich, but also rather complicated structure. In general, they are more difficult to understand compared to previous topologies. They have the following properties:

- Logarithmic diameter and constant vertex degree.
- Optimal or nearly optimal connectivity.
- Fault diameter nearly the same as non faulty one, similarly for fault average distances.
- Large bisection width of order $N/\log N$.
- Simple shortest-path routing.
- They are neither symmetric nor hierarchically recursive.
- They are rich of cycles of various lengths, since the rotation induces equivalence classes of vertices called ***necklaces***.
- At the same time, they are rich of subtrees and spanning trees, since the general shift operation induces subtrees.

The shuffle-based family includes three main topologies, shuffle exchange, de Bruijn, and Kautz, and their variations. We will describe only the first two main ones.

5.6.1 Shuffle-exchange networks

Even if the shuffle-exchange topology can be defined for any alphabet, the focus of computer science has always been primarily on the binary shuffle-exchange topology. The ***shuffle*** of a binary string x, $\sigma(x)$, is its ***left rotation by one bit***. The ***inversion of the least significant bit*** of x is called the ***exchange*** operation. ***Shuffle-exchange digraph of dimension*** n, SE_n, is defined as follows:

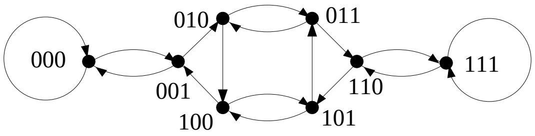


Figure 5.14. Digraph SE_3.

$$
\begin{array}{ll}
V(SE_n) = \mathcal{B}^n & |V(SE_n)| = 2^n \\
A(SE_n) = \{\langle x \rightarrow \mathrm{neg}_0(x)\rangle; x \in \mathcal{B}^n\} \cup \{\langle x \rightarrow \sigma(x)\rangle; x \in \mathcal{B}^n\} & |A(SE_n)| = 2^{n+1} \\
\deg^{\mathrm{in}}(SE_n) = \deg^{\mathrm{out}}(SE_n) = \{2\} & \phi(SE_n) = 2n-1 \\
\mathrm{bw_e}(SE_n) = \Theta(\frac{2^n}{n}) &
\end{array}
$$

- $\langle x \rightarrow \sigma(x)\rangle$ is called a ***shuffle arc*** and $\langle x \rightarrow \mathrm{neg}_0(x)\rangle$ is called an ***exchange arc***.
- The distance between vertices 0^n and 1^n is $2n-1$: we need n exchange and $n-1$ shuffle operations. The shortest dipath from vertex u to vertex v is constructed by finding the longest substring α such that $u = u'\alpha$ and $v = \alpha v'$ and then by traversing the shuffle and/or exchange arcs so that prefix u' is converted into postfix v'. We need ***at most*** $2n-1$ steps, which gives the diameter of SE_n.
- SE_n is a 2-regular digraph.
- The length of necklaces depends on the *periodicity* of strings. For example, if $n = 6$, then the necklace containing vertex 111010 has length 6, whereas the necklace containing vertex 101101 has length 3. In general, the necklaces can have length i where i divides n. Therefore, the neighborhood of a vertex in a SE digraph depends on its periodicity and the SE topology is *not* vertex symmetric.
- Due to this necklace structure, digraph SE_n does not contain as subgraphs any SE_i, $i < n$. Fortunately, there exists so called "approximate" hierarchical recursivity of shuffle-exchange topology, as we will explain in further sections.
- The exact value of $\mathrm{bw_e}(SE_n)$ is known up to a small constant multiplicative factor.

- Undirected shuffle-exchange topology is similar. Strings can be rotated to the right, which is so called ***unshuffle operation*** $\sigma^{-1}(x)$. The average distance drops, since unshuflle operation may provide shorter paths. For example, if $n = 10$, $u = 1011101010$, and $v = 1000010111$, the shortest directed path from u to v has length 13, whereas the shortest undirected path has length 8. The diameter, bisection width, the number and lengths of necklaces remain the same. The shuffle-exchange graph is not regular.

5.6.2 De Bruijn digraph of degree d and dimension n, $dB_{d,n}$

The adjacency is based on only one general operation of ***left shift*** by 1 position λ: If $x = x_{n-1} \ldots x_0 \in \mathcal{Z}_d^n$ and $a \in \mathcal{Z}_d$, then $\lambda(x, a) = x_{n-2} \ldots x_0 a$.

$$
\begin{array}{ll}
V(dB_{d,n}) = \mathcal{Z}_d^n & |V(dB_{d,n})| = d^n \\
E(dB_{d,n}) = \{\langle x \to \lambda(x,a)\rangle; x \in \mathcal{Z}_d^n, a \in \mathcal{Z}_d\} & |E(dB_{d,n})| = d^{n+1} \\
\deg^{\mathrm{in}}(dB_{d,n}) = \deg^{\mathrm{out}}(dB_{d,n}) = \{d\} & \phi(dB_{d,n}) = n \\
\mathrm{bw_e}(dB_{d,n}) = \Theta(d^n/n) &
\end{array}
$$

- Vertex u of $dB_{d,n}$ is adjacent to vertex v if the first $n-1$ letters of v equal to the last $n-1$ letters of u.
- Any vertex can be obtained from another one by at most n left shifts, hence the diameter. The fact that de Bruijn topologies have ***extremely low*** diameter is jeopardized by the fact that the ***average distance*** is very close to the diameter. This is a disadvantage compared to mesh-based topologies.
- $dB_{d,n}$ is d-regular, each vertex is incident to d and incident from d arcs.
- The structure of necklaces depends on periodicity of strings, similarly to shuffle-exchange topology, which destroys both the symmetry and the recursivity.
- Even though the topology is not recursive, the problem how to decompose $dB_{2,n}$ into isomorphic universal building blocks has been partially solved within project Galileo at JPL in Pasadena (they have built a parallel machine based on $dB_{2,16}$).
- Instead of hierarchical recursivity, de Bruijn topology has a different interesting property. $dB_{d,n}$ is the ***line digraph*** of $dB_{d,n-1}$, i.e., if we replace all arcs in $dB_{d,n-1}$ with vertices and preserve adjacency, we get $dB_{d,n}$. This property is used for optimal simulations of large de Bruijn networks on small ones.
- The connectivity is ***nearly*** optimal: $\kappa(dB_{d,n}) = \lambda(dB_{d,n}) = d-1$. It is only $d-1$ due to the loops at vertices a^n, $a \in \mathcal{Z}_d$ (see for example Figure 5.15).
- The fault diameter of $dB_{d,n}$ is only $n+1$: The lengths of all $d-1$ vertex-disjoint paths joining two arbitrary vertices u and v are at most $n+1$.
- The exact value of $\mathrm{bw_e}(dB_{d,n})$ is known up to a small constant multiplicative factor.
- The binary de Bruijn digraph $dB_{2,n}$ is the state diagram of a shift register. It needs only 4 links per node to achieve the same diameter as equally sized hypercube.
- Besides digraphs, there exist also undirected graph versions. Most properties of digraphs apply to graphs, including the complexity of communication algorithms.
- Also the problem of incremental extendability and scalability was partially solved. There exist modified de Bruijn topologies, called ***generalized*** de Bruijn networks, which are incrementally extendable, and so called ***partial line digraphs***, which are even incrementally scalable. However, the topological complexity of these variations is even higher than that of standard ones.

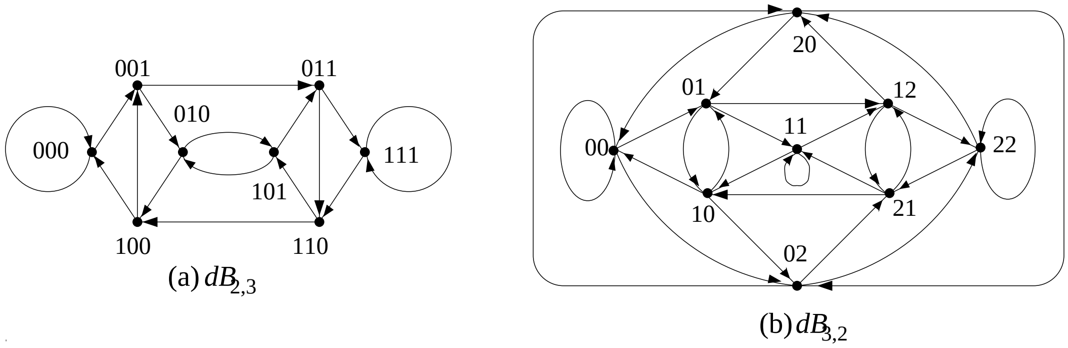


Figure 5.15. De Bruijn digraphs $dB_{2,3}$ and $dB_{3,2}$.