**Question**

•Explain the following Interconnection networks:

1.Torus

2.Hypercube Interconnection Network

[Hypercube](https://en.wikipedia.org/wiki/Hypercube) networks are a type of [network topology](https://en.wikipedia.org/wiki/Network_topology) used to connect multiple [processors](https://en.wikipedia.org/wiki/Processors) with memory modules and accurately route data. Hypercube networks consist of 2m nodes. These nodes form the vertices of squares to create an internetwork connection. A hypercube is basically a multidimensional [mesh network](https://en.wikipedia.org/wiki/Mesh_networking) with two nodes in each dimension. Due to similarity, such topologies are usually grouped into a k-ary d-dimensional mesh topology family where d represents the number of dimensions and k represents the number of nodes in each dimension.

Hypercube interconnection network is formed by connecting N nodes that can be expressed as a power of 2. This means if the network has n nodes it can be expressed as :

{\displaystyle N=2^{m}}where m is the number of bits that are required to label the [nodes](https://en.wikipedia.org/wiki/Mesh_node) in the network. So, if there are 4 nodes in the network, 2 bits are needed to represent all the nodes in the [network](https://en.wikipedia.org/wiki/Computer_network). The network is constructed by connecting the nodes that just differ by one bit in their [binary](https://en.wikipedia.org/wiki/Binary_code) representation. This is commonly referred to as Binary labelling. A 3D hypercube internetwork would be a cube with 8 nodes and 12 [edges](https://en.wikipedia.org/wiki/Edge_(geometry)). A 4D hypercube network can be created by duplicating two [3D](https://en.wikipedia.org/wiki/Three-dimensional_space) networks, and adding a most significant bit. The new added bit should be ‘0’ for one 3D hypercube and ‘1’ for the other 3D hypercube. The corners of the respective one-bit changed [MSBs](https://en.wikipedia.org/wiki/Most_significant_bit) are connected to create the higher hypercube network. This method can be used to construct any m-bit represented hypercube with (m-1)-bit represented hypercube.

E-Cube Routing

Routing method for a hypercube network is referred to as E-Cube routing. The distance between two nodes in the network can be given by [Hamming weight](https://en.wikipedia.org/wiki/Hamming_weight) of (number of ones in) the [XOR](https://en.wikipedia.org/wiki/Exclusive_or)-operation between their respective binary labels.

The distance between Node 1 (represented as ‘01’) and Node 2 (represented as ‘10’) in the network given by:

({\displaystyle Hamming\\_weight(01\bigoplus 10)=Hamming\\_weight(11)=2}

E-Cube routing is a [static routing](https://en.wikipedia.org/wiki/Static_routing) method that employs XY-routing [algorithm](https://en.wikipedia.org/wiki/Algorithm). This is commonly referred to as [Deterministic](https://en.wikipedia.org/wiki/Deterministic_algorithm), [Dimension](https://en.wikipedia.org/wiki/Dimension) Ordered [Routing](https://en.wikipedia.org/wiki/Routing) model. E-Cube routing works by traversing the network in the kth dimension where k is the least significant non-zero bit in the result of calculating distance.

For example, let the sender's label be ‘00’ and the receiver's label be ‘11’. So, the distance between them is 11 and the least significant non-zero bit is the [LSB](https://en.wikipedia.org/wiki/Least_significant_bit) bit. Figuring out which way to go for a ‘0’ or ‘1’ is determined by XY routing algorithm.

Metrics

Different measures of performance are used to evaluate the efficiency of a hypercube network connection against various other network topologies.

**Degree**

This defines the number of immediately adjacent nodes to a particular node. These nodes should be immediate neighbors. In case of a hypercube the degree is n.

**Diameter**

This defines the maximum number of nodes that a message must pass through on its way from the source to the destination. This basically gives us the delay in transmitting a message across a network. In case of a hypercube the diameter is n.

**Average Distance**

The distance between two nodes defined by the number of hops in the shortest path between two particular nodes. It is given by the formula -

{\displaystyle d\_{a}=\sum \_{d=1}^{r}{(d.N\_{d}) \over N-1}}

In case of Hypercubes the average distance is given as n/2.

**Bisection Width**

This is the least number of wires that you should cut in order to divide the network into two equal halves. It is given as 2n-1 for Hypercubes.

The hypercube is one of the most popular static topologies. A hypercube of dimension n, called an n-cube, consists of 2n nodes, where each node represents a processor (with its local memory). The processors are numbered from 0 to 2n 1 and are given an n-bit binary address corresponding to their number. Thus processor 3 in a 3-cube is given the address 011. Two processors are connected by a link if their binary addresses di er in exactly one bit position. The link connecting processors with addresses di ering in the i-th bit position is called the i-th dimension link. For example, when n = 3, the processor with address 010 is connected to the processors 011, 000 and 110. The Figure illustrates hypercubes of dimensions 2 and dimensions 3. Hypercubes can be dened recursively; an (n + 1)-cube is constructed from two n-cubes by (1) prexing the addresses of processors in one hypercube with a 0 (to get the zero cube) and the addresses in the other with a 1 (to get the one cube) and (2) each of the processors in the zero cube is connected to its counterpart in the one cube. The distance between any two processors is dened by the hamming distance between their addresses, and thus we see that the maximum distance between any two processors (i.e., the diameter) is n for an n-cube which has 2n processors.

Point-to-point routing: on the hypercube can be performed by using a simple algorithm. The shortest path between any two processors can be determined by performing an exclusive OR of their addresses. This leads to a simple routing policy; perform an exclusive OR of the source address A and destination address B and route along dimension i if the i-th bit of the exclusive

OR operation is a 1. For example, consider the transfer of data from processor 010 to processor 111. The exclusive OR gives 101 and thus we need to route along dimension 0 link and dimension 2 link. This results in a data transfer from 010 to 011 in the rst step and from 011 to 111 in the second step. In general, the routing algorithm can be designed to work in a specic dimension sequence; for example it could send along dimension 0 rst and then dimension 1 etc. etc.

One-to-All: Broadcasting in a hypercube is accomplished by exploiting the recursive structure of the hypercube. The processor with the message rst sends a message to one of its neighbors. At the second step both processors send to neighbors in one dimension and this process is repeated. Essentially, a broadcast is performed in cubes of dimension 0, 1; :::; i; i + 1; :::.

Torus network

The torus network router directs variable-size packets, each n 3 32 bytes, where n = 1 to 8 ‘‘chunks.’’ Messages, such as those conforming to the Message Passing Interface Standard (MPI), may consist of many packets that are constructed, sent, and received by software running on one or both associated BG/L processors. The first eight bytes of each packet contain link-level protocol information (e.g., sequence number); routing information, including destination; virtual channel and size; and a byte-wide cyclic redundancy check (CRC) that detects header data corruption during transmission. In addition, a 24-bit CRC is appended to each packet, along with a one-byte valid indicator. The valid indicator is necessary, since packets can be forwarded before being entirely received. This CRC permits checking of each packet as it is sent over each link. A time-out mechanism is used for retransmission of corrupted packets. Use of the eight-bit packet header CRC is an optimization that permits early detection of packet header errors because the header CRC is included in the full packet CRC. The error detection and recovery protocol is similar to that used in IBM High Performance Switch no routing tables. Packets may be either dynamically or deterministically dimension-ordered (xyz) routed. That is, they can follow a path of least congestion based on other traffic, or they can be routed on a fixed path. Besides point-to-point packets, a bit in the header may be set that causes a packet to be broadcast down any Cartesian dimension and deposited at each node. Hardware broadcasting in more than one dimension would have greatly complicated the logic, although deadlocks could have been prevented by broadcasting in the same xyz order, as described in [5]. The hardware does not have the capability to route around ‘‘dead’’ nodes or links. However, software can set the hint bits appropriately so that such nodes are avoided; full connectivity can be maintained when there are up to three noncolinear faulty nodes. The torus logic consists of three major units—a processorinterface, a send unit, and a receive unit (Figure 1). The processor interface consists of network injection and reception FIFOs (queues in which access is according to the first in, first out rule). Access to these FIFOs is via the double floating-point unit (FPU) registers; i.e., data is loaded into the FIFOs via 128-bit memory-mapped stores from a pair of FPU registers, and data is read from the FIFOs via 128-bit loads to the FPU registers. There are a total of eight injection FIFOs organized into two groups: two high-priority (for internode operating system messages) and six normal-priority FIFOs, which are sufficient for nearest-neighbor connectivity. Packets in all FIFOs can go out in any direction. On the reception side, there are again two groups of FIFOs. Each group contains seven FIFOs, one highpriority and one dedicated to each of the incoming directions. More specifically, there is a dedicated bus between each receiver and its corresponding reception FIFO. As in mechanisms in the BG/L collective network, there is a checksum associated with each injection FIFO; these can be used for fault-isolation and debugging purposes to see whether a node sends the same data onto the network in two different runs. Watermarks can also be set by software for the injection and reception FIFOs so that interrupts fire when the FIFO contents cross the corresponding threshold. For storage, all torus FIFOs use static random access memory chips (SRAMs) protected by error checking and correction (ECC), and all internal data paths are checked for parity. There is a total of 58 KB of SRAMs. The datapath for each of the six receivers, is composed of an eight-stage input pipeline, four virtual channels (VCs), and a bypass channel. The input pipeline processes the packet header on the fly. Multiple VCs help reduce head-of-line blocking [6], but—since mesh networks, including tori with dynamic routing, can deadlock—appropriate additional escape simultaneously; thus, each node can be sending and receiving 1.05 GB/s. In addition, there are two transfer buses (paths) coming out of each receiver that connect with the senders. As a result, a single receiver can have up to four simultaneous transfers, e.g., one to its normal processor reception FIFO, one to the high-priority processor reception FIFO, and two to two different senders. The torus network input and output are bytewide and are serialized and deserialized by the high-speed signaling units outside the torus module. The torus internal buses are all 11 bits wide (eight data, one parity, two control). Arbitration is distributed and pipelined, but occurs in three basic phases. It generalizes an approach described in [11] and represents tradeoffs among complexity, performance, and the ability to meet timing constraints. In the first phase, a decision is made for each packet at the head of the injection or VC FIFOs about the direction in which it should preferably move and which VC it should use. There is only one valid choice for deterministically routed packets, but there may be many choices for dynamically routed packets. The preferred direction and VC are selected using a modified join-the-shortest-queue (JSQ) algorithm, as follows. The senders provide the receivers and injection FIFOs with a bit that indicates both link and token availability for each VC in each direction. This bit vector is ANDed with a bit vector of possible moves constructed from the packet hint bits and VC. This defines the set of possible and available arbitration requests. In addition, for each VC the sender provides two bits that indicate one of four ranges of available downstream tokens. Of all the possible and available dynamic direction and VC pairs, the packet selects the one with the most available downstream tokens. Ties are randomly broken. If no combination of dynamic direction and VC is available, the packet requests its bubble escape direction and VC pair (if available). If that is also unavailable, no arbitration request is made for the packet. This is a somewhat simplified description, since data bus availability must also be taken into account. In addition, when a packet reaches its destination, the ‘‘direction’’ requested is simply the corresponding reception FIFO. In the second phase, arbitration is required to determine which of the requesting packets in the receiver wins the right to request, since each receiver has multiple VC FIFOs in addition to the bypass channel. If a highpriority packet is requesting, it wins. Barring that, a modified serve the longest queue (SLQ) is used, based on two-bit (four ranges) FIFO fullness indicators; i.e., the packet from the VC that is fullest as measured to within the two bits of granularity wins. However, this algorithm cannot always be used, because doing so may completely block out a VC. Therefore, a certain (programmable) destination and size of the message, and so on. This pseudocode included a subset of the message passing interface (MPI) point-to-point messaging calls as a workload driver for the simulator. We also extended the IBM unified trace environment trace capture utility that runs on IBM SP\* supercomputer machines and were able to use such traces as simulator input for up to several hundreds of nodes. The basic unit of simulation time is a network cycle, defined to be the time it takes to transfer one byte. Because the BG/L is organized around 512-node (83838) midplanes, the simulator partitions its work on a midplane basis; i.e., all nodes on the same midplane are simulated by the same processor thread, and midplanes are assigned to threads in as even a manner as possible. Because different threads are concurrently executing, the local simulation clocks of the threads must be properly synchronized. To deal with this problem, we use a simple but effective ‘‘conservative’’ parallel simulation protocol known as YAWNS (yet another windowing network simulator) [15]. In particular, we take advantage of the fact that the minimum transit time between midplanes is known and is at least some constant w 1 cycles. In this protocol, time ‘‘windows’’ of length w are simulated in parallel by each of the threads. At the end of each window, a synchronization phase takes place in which each thread picks up the information about packets that will arrive in future windows. If w = 1, this protocol requires a barrier synchronization every cycle. On BG/L, however, the minimum inter-midplane delay is approximately w = 10 network cycles. When a large number of BG/L nodes are being simulated, each processor executes many events during a window, i.e., between barriers; thus, the simulator should obtain good speedups. The simulator runs on a 16-way IBM POWER3þ\* 375-MHz symmetric multiprocessor (SMP) node with 64 GB of memory. The model of the torus hardware contains close to 100 resources per node (links, VC token counters, buses, FIFOs, etc.), so that a full 64K-node system can be thought of as a large queuing network with approximately six million resources. The model consumes a large amount of memory and runs slowly; a 32K-node simulation of a fully loaded network advances at about 0.25 microseconds of BG/L time per second of wall clock time. However, it obtains excellent speedup—typically, more than 12 on 16 processors—and sometimes achieves superlinear speedup because of the private 8-MB L3 caches on the SMP and the smaller per-node memory footprint of the parallel simulator. The model, which was written before the hardware design in Very high-speed integrated circuit Hardware Description Language (VHDL) was implemented, is thought to be a quite accurate representation of the BG/L