In Proceedings of 12th IASTED International Conference on Parallel and Distributed Computing and Systems

k-ary *m*-way Networks: the Dual of *k*-ary *n*-cubes

Muhammed F. Mudawwar

Computer Science Department
The American University in Cairo
113 Kasr el Aini street, Cairo, Egypt
mudawwar@aucegypt.edu

Abstract

This article presents *k*-ary *m*-way networks, multi-dimensional mesh and tori networks that are viewed as the dual of *k*-ary *n*-cube networks. An *m*-way channel is the physical wiring of *m* router and processor links. An *m*-way router interfaces two channels only, irrespective of the network topology or dimension. This has important advantages: the same router can be used to build networks of different dimensionalities or topologies, physical channels can be very wide, and broadcasting and multicasting are facilitated. Routing in a *k*-ary *m*-way network is detailed in this paper. The performance of *k*-ary *m*-way meshes, tori, and hypercubes is evaluated for different routing algorithms.

Keywords: *k*-ary *m*-way network, *m*-way channel, *m*-way router, bus-based interconnection network, performance evaluation.

1 Introduction

An interconnection network is often a critical part of a massively parallel computer because application performance is sensitive to network latency and throughput. An important class of interconnection networks is the direct network, in which routers and nodes are linked directly using dedicated point-to-point channels. *k*-ary *n*-cubes are strictly orthogonal direct topologies with *n* dimensions and *k* routers (nodes) along each dimension. Low dimensional *k*-ary *n*-cube networks have been implemented in many parallel architectures, which include the Intel Teraflops [3], MIT J-Machine [10], and Cray T3E [11].

A physical bi-directional link connecting two routers in a *k*-ary *n*-cube network can be implemented either as one set of bi-directional wires called half-duplex organization, or as two sets of unidirectional wires called full-duplex organization. With a full-duplex organization, a router element has 4*n* input and output channels to adjacent routers. As the dimensionality, *n*, of a network increases, the number of input and output channels also increases. Since the number of I/O pins in a router chip is limited by the packaging technology, the increase in the dimensionality of a network will decrease the number of wires and thus the bandwidth of a single physical channel. Routers designed for low dimensional *k*-ary *n*-cube networks have physical channels that typically are 8-bit-data to 16-bit-data wide.

In this study, I am proposing a new class of interconnection networks, called *k*-ary *m*-way networks that can be viewed as the dual of *k*-ary *n*-cubes. The idea is to exchange routers and channels in a *k*-ary *n*-cube network. Routers in a *k*-ary *n*-cube are replaced with *m*-way channels in a *k*-ary *m*-way network, while bi-directional links in a *k*-ary *n*-cube are replaced with *m*-way routers. This will be detailed in the next section.

2 *k*-ary *m*-way Networks

An *m*-way (called also *multiway*) channel is a physical channel *shared* by a maximum number, *m*, of routers or processors. It is the physical wiring of *m* links. An *m*-way router interfaces two *m*-way channels only, irrespective of the network topology or dimension. It has a constant degree 2. An *m*-way router defines the operation of an *m*-way channel. At any clock cycle, only one of the *m* routers (or processors) linked to an *m*-way channel can drive the channel. However, all *m* routers (and processors) can concurrently read the channel. *m*-way routers, of constant degree 2, can be used to define a variety of network topologies.

A *k*-ary *m*-way network is a multi-dimensional mesh or torus structure constructed using *m*-way routers and channels. The factor *k* is the number of *m*-way channels along each dimension. To simplify equations, one factor *k* is used for all dimensions, but in practice different values of *k* can be assigned to different dimensions. The maximum number of ways, *m*, of an *m*-way channel is called the *sharing factor* of a channel.

There are two approaches of linking processors (their local memories and/or caches) to a *k*-ary *m*-way network. The first approach is to link processors to channels directly. A 3-ary 5-way torus with one processor linked to each channel is shown in Figure 1. A channel is identified as c_i, a processor node linked to channel c_i is identified as P_i, and a router linked to channels c_i is identified as R_{ix} if it along the positive X dimension, or R_{iy} if it is along the positive Y dimension. Although one processor is shown connected to each channel, it is possible to link several ones. If *p*

processors are linked to each m-way channel and n is the network dimension then $m = 2n + p$. p is called the *processor factor*. The second approach is to have processors linked internally to routers. Figure 2 shows a 3-ary 4-way torus with one processor linked internally to each router. Here, a processor node has the same identification of its associated router. The sharing factor $m = 2n$ in this case.

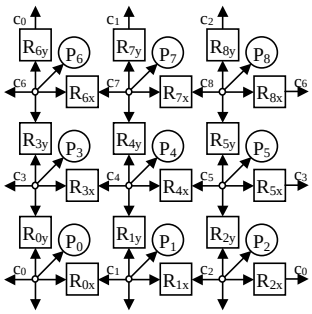

Figure 1: A 3-ary 5-way torus processors linked to channels

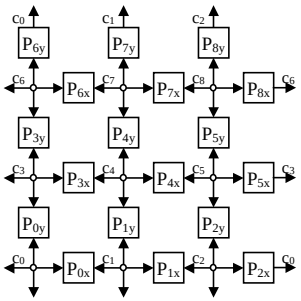

Figure 2: A 3-ary 4-way torus processors linked to routers

Linking processors directly to channels simplifies routing and the router design, but will increase the sharing factor of an m-way channel. The number of processors can be less than, equal to, or greater than the number of routers, depending on the processor factor. On the other hand, integrating a processor within each router has the advantage of reducing the sharing factor, m, and the total number of components of a system. However, routing and the router design will be more complicated.

2.1 Topology

A k-ary $2n$-way network ($m = 2n$) is defined recursively as $2k$ k-ary $(2n-2)$-way networks wired together orthogonally to produce the nth dimension. The wiring is done on channels, rather than on routers. A k-ary 2-way network is either a linear array of routers or a ring. If linear arrays of routers are used then the resulting k-ary $2n$-way network will have a mesh structure. If rings are used, the resulting k-ary $2n$-way network will have a torus structure. If p processors are linked directly to each channel then this will result in a k-ary $(2n+p)$-way network ($m = 2n + p$).

A channel address, c, is an n-digit radix k number: $c = a_{n-1}\ldots a_i\ldots a_0$. Each digit a_i represents a channel's coordinate in the i^{th} dimension and can take the values 0 through k-1. Two m-way channels are adjacent if their addresses differ in one digit by $\pm$ 1 in a mesh and $\pm$ 1 mod k in a torus network. Between every 2 adjacent channels is an m-way router. A router linked to channel $a_{n-1}\ldots a_i\ldots a_0$ has address $a_{n-1}\ldots a_i\ldots a_0$, i if it is along the positive i^{th} dimension, or address $a_{n-1}\ldots(a_i-1 \bmod k)\ldots a_0$, i if it is along the negative i^{th} dimension. The modulo-k operation is necessary when the network has a torus structure. A processor linked to channel $a_{n-1}\ldots a_0$ has address $a_{n-1}\ldots a_0$, l where l can assume the values 0 through $p-1$. If $p = 1$ then a processor address becomes equal to its corresponding channel address. A processor linked internally to a router will assume the router address.

2.2 Properties

Some properties of k-ary m-way networks are shown in Figure 3 and are contrasted with the properties of k-ary n-cube networks. Four k-ary m-way network types are considered to distinguish a torus from a mesh and whether processors are linked directly to channels ($m = 2n + p$) or internally to routers ($m = 2n$). For k-ary n- cube networks, full duplex bi-directional channels are assumed between adjacent routers.

Network Type			Channels	Routers	Processors	Diameter	Degree
k-ary m-way	Torus	$m = 2n + p$	k^n	$n\,k^n$	$p\,k^n$	$n\lfloor k/2 \rfloor+1$	2
		$m = 2n$	k^n	$n\,k^n$	$n\,k^n$	$\lceil n\,k/2 \rceil$	2
	Mesh	$m = 2n + p$	k^n	$n\,(k-1)\,k^{n-1}$	$p\,k^n$	$n\,(k-1)+1$	2
		$m = 2n$	k^n	$n\,(k-1)\,k^{n-1}$	$n\,(k-1)\,k^{n-1}$	$n\,(k-1)-1$	2
k-ary n-cube	Torus		$2\,n\,k^n$	k^n	k^n	$n\lfloor k/2 \rfloor$	$4\,n$
	Mesh		$2n\,(k-1)\,k^{n-1}$	k^n	k^n	$n\,(k-1)$	$4\,n$

Figure 3: Some properties of k-ary m-way and k-ary n-cube networks

A k-ary m-way network has several advantages. First, an m-way channel can be made much wider than a direct channel used in a k-ary n-cube network. For example, an 8-way router design for low-dimensional k-ary m-way networks, given in [9], defines 8-way channels with 128 data and 17 control lines. Thus, 290 I/O pins are used for both channel interfaces in a router. This is to be contrasted with the router chip of the Cray T3D, which uses 16 data lines and 8 control lines per physical channel. There are 6 input and 6 output channels for a total of 288 I/O pins, not counting the lines connecting to the local node. Therefore, although there are more channels in a k-ary n-cube than in a k-ary m-way network of similar size and cost as shown in Figure 3, each channel in a k-ary m-way network can be made much wider. This example also shows that the overhead of control lines can be less in a k-ary m-way network.

A second advantage of k-ary m-way networks and m-way routers is that the same router, if carefully designed, can be used to implement networks of different dimensionalities. For instance, an m-way router can be used to implement various dimensional meshes and tori. This is more difficult to achieve with direct channel routers because the dimensionality of a network is related to the number of links per router. A third advantage is that m-way channels facilitate broadcasting and multicasting because they are

shared. A message flit placed on a channel can be accepted in multiple routers concurrently during the same clock cycle.

2.3 Graph Model

A direct interconnection network is modeled as a strongly connected directed multigraph $I_d = G(N, C)$ [6]. The vertices of I_d are the network nodes, denoted as set N. The arcs are the virtual channels (or buffers), denoted as set C. More than a single virtual channel is allowed to exist between two adjacent nodes, hence it is called a multigraph. Although this graph model is useful for direct networks, it is not appropriate for m-way-channel based networks. Therefore, a new graph model is needed.

DEFINITION 1: An m-way-channel interconnection network is modeled as a strongly connected directed multigraph $I_m = G(P \cup C , B)$. The vertices of I_m are the network processors, P, and the m-way channels, C, denoted as the union set $P \cup C$. The arcs are the network buffers, denoted as the set B. Buffers exist in routers and in processor interfaces. A router interfacing channels c_i and $c_j \in C$ will have buffers from c_i to c_j, identified as ordered triples (c_i , c_j , #), where # is used to number the individual buffers. There are also buffers from c_j to c_i, identified as (c_j , c_i , #). A processor $p_i \in P$ interfacing a channel $c_j \in C$ will have injection buffers, identified as (p_i , c_j , #), as well as ejection buffers, identified as (c_j , p_i , #). Therefore, $B \subset (C \times C \times \mathbf{N}) \cup (P \times C \times \mathbf{N}) \cup (C \times P \times \mathbf{N})$, where $\mathbf{N}$ is the set of non-negative integers.

The graph model of the 3-ary 5-way torus network of Figure 1 is shown in Figure 4. The processors and channels are the vertices of this graph. The buffers are the arcs. There is one injection and one ejection buffer at the interface of each processor, and two buffers for each routing direction. A message starts at an injection buffer of a source processor. It uses a sequence of buffers in the network graph until it reaches an ejection buffer of a destination processor. Messages are buffered in the arcs of a network graph. This view is consistent with the graph model of a direct network, in which arcs are virtual channels, or buffers. This means that the routing algorithms and the deadlock-avoidance theories discussed in the literature [6], [8] for wormhole-routed k-ary n-cube networks are also applicable to k-ary m-way networks.

2.4 Related Work

An m-way channel is a bus and a k-ary m-way network can be classified as a bus-based interconnection network. Many bus interconnection structures were discussed in the literature. They are modeled as *hypergraphs* [1]. A hypergraph is a set of vertices and a set of hyperedges. The vertices are the processor nodes. The hyperedges, identified as subsets of vertices, are the buses. A hypergraph does not identify the buffer resources of a network. Hence, it is not a useful tool to study routing algorithms and deadlocks in a k-ary m-way network.

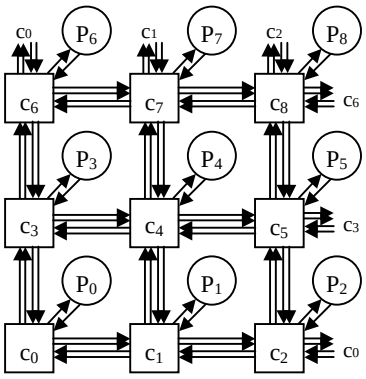

Figure 4: Graph model of a 3-ary 5-way torus

Examples of bus interconnection networks are the hypermesh [12], hypergrid (hypertorus) [7], and hyperbus [2]. In a hypermesh, each node is connected to all the nodes in each dimension through a bus. If k is the number of nodes along each of the n dimensions then nk is the number of buses in the network. Each node is connected to n buses and the network diameter is n. The hypergrid and hypertorus structures are defined as the Cartesian product of hyperpaths and hyperrings [7]. The node degree is not a constant, but is twice the network dimension. The hyperbus is defined as the dual of a generalized hypercube. Each node is connected to exactly two buses, but the network topology is different than a k-ary m-way network. To minimize distances between nodes, bus interconnection networks tend to have a large number of nodes sharing a small number of buses. The sharing factor of each bus tends to be high. This increases the length of wires, the system packaging costs, and reduces the speed of buses. On the other hand, an m-way channel in a k-ary m-way network is localized and shared by a small number of routers and processors. This localization is meant to minimize the length of wires, to reduce the system packaging costs, and to increase the speed of shared channels.

3 Routing

Routing in a k-ary m-way network is different than routing in a k-ary n-cube network. In a k-ary n-cube network, a router routes a header flit internally from an input buffer to an output buffer. The routing logic decides which output channel (physical and virtual) to select according to a routing function and a selection function. The routing logic controls a crossbar switch that establishes simultaneous paths between input and output buffers.

Routing in a *k*-ary *m*-way network is to determine the next router and buffer along a routing path. The routing logic determines the *routing way* and the *buffer class*, denoted as (*way*, *class*), according to a routing function and a selection function. The *way* specifies the next router across an *m*-way channel. The *class* specifies a subset of buffers that can be allocated when a header flit is received. The buffer *class* ensures deadlock freedom for some routing algorithms. The routing function specifies one (deterministic) or more (adaptive) choices of (*way*, *class*) pairs, and the selection function chooses one of them (in case of adaptive routing). The routing function must be deadlock-free and livelock-free. The selection function can affect only performance.

Once a (*way*, *class*) pair is determined, a header flit is ready to be transferred. When a channel driver places a header flit on an *m*-way channel, it includes the routing *way* and the buffer *class* as part of the header information. All routers and processor interfaces examine the header flit, but only one accepts it according to *way*. Once a header flit is accepted, the allocation unit at the receiver side allocates a free buffer in the specified buffer *class*. This buffer is kept allocated for all the flits of a message.

3.1 Formal Definitions

Given the following set definitions:

Buffer = set of all buffers in a *k*-ary *m*-way network (same as *B* in Definition 1)
Channel = set of all *m*-way channels (same as *C* in Definition 1)
Processor = set of all processors (same as *P* in Definition 1)
Way = set of all routing ways = {X+, X–, Y+, Y–, etc.}
Class = set of all buffer classes = {*adaptive*, *deterministic*, etc.}

Routing of a header flit in a *k*-ary *m*-way network can be described with four functions: *drive*, *route*, *select*, and *allocate*.

DEFINITION 2: *drive*: *Buffer* → *Channel* is a function that maps a buffer $b \in$ *Buffer* to an *m*-way channel $c \in$ *Channel*. It means that buffer *b* drives channel *c*. *b* can be an injection buffer or a router buffer, but cannot be an ejection buffer. If *b* is an ejection buffer then an error result is returned.

DEFINITION 3: *route*: *Channel* × *Processor* → P (*Way* × *Class*) is a function that returns a set of (*way*, *class*) pairs that identifies all the next buffers that can accept a header flit along the routing paths from a current channel $c \in$ *Channel* to a destination processor $p \in$ *Processor*. P ( ) is the power set.

DEFINITION 4: *select*: P (*Way* × *Class*) × *Channel* × *avail* → *Way* × *Class* is a function that returns one (*way*, *class*) pair that can receive a header flit from a channel $c \in$ *Channel*. The decision is based on an *avail* function: *Channel* × *Way* × *Class* → **N** that returns the number of available buffers in all the routing ways and buffer classes that can be reached from a channel *c*. The input set of (*way*, *class*) pairs should not be empty; otherwise, *select* returns an error result.

DEFINITION 5: *allocate*: *Channel* × *Way* × *Class* → *Buffer* returns a free buffer $b \in$ *Buffer* in the buffer *class* $\in$ *Class* that can be reached from a channel $c \in$ *Channel* along the routing *way* $\in$ *Way*. If no buffer is found free, *allocate* returns an error result.

3.2 Router Structure

A router for *k*-ary *m*-way networks is depicted in Figure 5. A router has two channel interfaces, two channel arbitrators, and two groups of buffers with allocation/mapping units, routing logic, and buffer arbitrators. The *directionalities* of the two groups of buffers are DIM+ and DIM–, where DIM is the dimensionality of the router. The directionality is used to identify a buffer set when selecting a driver for a channel or when accepting message flits. This identification should be unique across an *m*-way channel. Observe that no crossbar switch is required. This simplifies the implementation of a router and makes it faster.

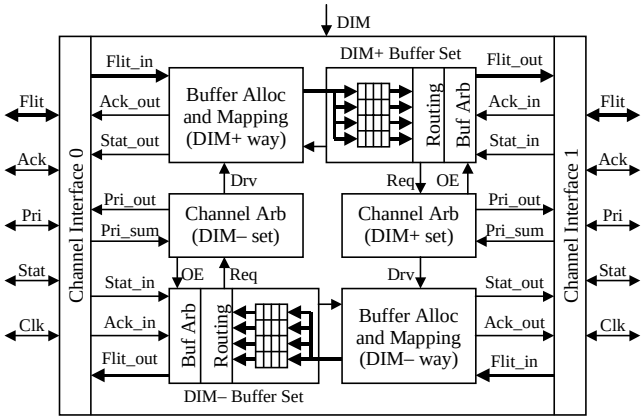

Figure 5: Internal Structure of a Router

The router structure of Figure 5 assumes that processor nodes are linked directly to channels. If processors are linked internally to routers then the router structure has to be modified to include injection and ejection buffers in both directions. For the remaining of this paper, I consider only processor nodes linked directly to channels as shown in Figure 1.

A physical channel consists of data, control, and arbitration lines. The *Flit* lines carry one flit of a message. The *Ack* lines are used to acknowledge the transfer of a flit and to report the full status of the receiver buffer. The priority

lines, *Pri*, are used for arbitration and carry the wired-OR sum of output priorities of requesting drivers. The *Stat* lines carry the availability and full status of receiver buffers. The *Clk* line is used to synchronize the operation of an *m*-way channel.

3.3 Buffer Status and Flow Control

Each buffer in a buffer set has associated status information, as depicted in Figure 6. The allocation bit, *A*, indicates the allocation status. The full bit, *F*, indicates the full status. The driver number, *Drv*, indicates the driver from which the flits of a message are received. The driver's buffer number, *Buf*, indicates from which buffer a flit is received. *Drv* and *Buf* locate the previous buffer along the routing path. The front pointer, *Fptr*, points to the front entry in a buffer. The rear pointer, *Rptr*, points to the rear entry. The receiver's full status, *RF*, indicates whether the receiver buffer of a message has a full status.

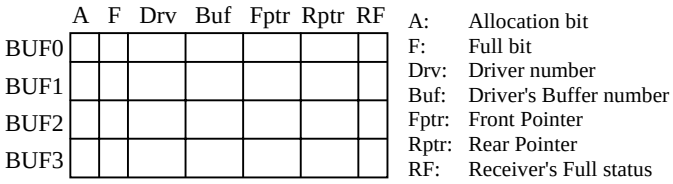

Figure 6: Buffer status information

The flow control mechanism of a network determines how buffers are allocated and freed. The allocation must be done in a manner that keeps the flits associated with a particular message together. When a header flit is received, a buffer is allocated (Allocation bit *A* is set). An allocated buffer is freed after a tail flit is transmitted and the buffer is emptied.

3.4 Router Operation

An *m*-way channel can have only one router or processor driving it at any given clock cycle. The channel arbitrator ensures exclusive access to the channel. It determines which router (processor) is driving a multiway channel at the current clock cycle and which router (processor) will be driving the channel at the next cycle. Channel arbitration is a distributed hardware algorithm. All channel drivers apply the same algorithm and reach the same decision. The channel arbitrator must be fair to avoid starvation. An implementation for 8-way channels that uses Round Robin with 3 priority lines is discussed in [9].

At the beginning of a clock cycle, a driver puts a header flit on an *m*-way channel. The header flit carries the header tag, *H*, the driver's buffer number, *buf*, and the routing *way* in addition to other control information. The buffer allocation and mapping units in all the directions of an *m*-way channel examine the header flit. However, only one buffer allocation unit will accept the header flit, depending on the routing *way*. Once accepted, the buffer allocation unit will allocate a buffer for the header flit and send back an acknowledgment, *ack_out*.

When a driver places a body or a tail flit on a channel (*tag* = *B* or *T*), it does not include the routing *way* as part of the flit. All allocation and mapping units across a channel examine the flit that carries the driver's buffer number, *buf*. They also obtain the current driver number, *drv*, from the channel arbitrator. All allocation and mapping units are searched by content for a match with *drv* and *buf*. If a match occurs and the allocation bit is set., the corresponding buffer allocation and mapping unit will accept the body or tail flit. Otherwise, it will reject. Once accepted, an acknowledgment is sent back.

3.5 Routing Algorithms

A *k*-ary *m*-way network with shared channels can use the same routing algorithms developed for a *k*-ary *n*-cube network with direct channels. Four routing algorithms are used for the simulation of *k*-ary *m*-way networks in the next section. The first algorithm is dimension-order routing (DOR) [4]. This algorithm is known to prevent deadlocks in mesh and hypercube topologies because it does not allow cycles in the channel dependency graph. No buffer classes are required and any buffer can be allocated when a header flit is received. We can also apply the early buffer free policy and allow more than one message to be queued in a single buffer without causing deadlocks.

Dimension-order routing is, however, problematic in the case of a torus. We need to avoid deadlocks caused by the rings along all dimensions. An efficient routing algorithm that avoids deadlocks in a unidirectional ring and makes good use of buffers is presented in [5]. This algorithm divides buffers in two classes *low* and *high*. We allocate a buffer in the *low* class if the destination node address is *less than* the current node address. We allocate a buffer in *any* class if the destination node address is *greater than* the current node address. This algorithm was shown to avoid deadlocks. Although cycles exist in the channel dependency graph according to [4], no cycles exist in the new extended channel dependency graph according to [6].

The ring algorithm used in this paper is a minor modification to the one presented in [5]. It also uses two buffer classes: *low* and *high*. The buffer classes are used for router buffers only. They are not required for injection and ejection buffers. An example of an 8-ary 3-way ring with 2 router buffers along each direction and 1 injection/ejection buffer at the interface of each processor node is shown in Figure 7. The channels and processors are divided into two groups. Group 0 consists of c_0 to c_3 and P_0 to P_3. Group 1 consists of c_4 to c_7 and P_4 to P_7. A buffer that drives a

channel in Group 0 belongs to group 0; otherwise, it belongs to Group 1. The ring algorithm is defined as follows: if the next buffer along the routing path and the destination processor belong to the same group the next buffer can be allocated from *any* class. If they belong to different groups, the next buffer can be allocated only from the *low* class. This algorithm can be shown to avoid deadlocks according to the theory presented in [6].

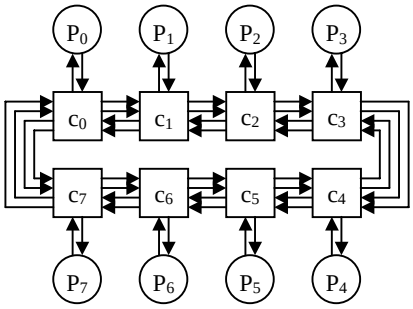

Figure 7: Graph of an 8-ary 3-way Ring

The ring algorithm makes adaptive decisions when the distance between a source and a destination processor is the same along the positive and negative directions in a given dimension. The DOR algorithm can be combined with the ring algorithm to obtain a partially adaptive deadlock-free minimal algorithm for tori networks, referred to as DOR_RING. The DOR_RING algorithm requires 2 router buffers per buffer set, irrespective of the number of dimensions. Adding more buffers improves performance, but is not required to avoid deadlocks.

A fully adaptive deadlock-free minimal routing algorithm for *k*-ary *m*-way mesh and hypercube networks can be designed as follows: Two buffer classes, *deterministic* and *adaptive*, are required irrespective of the number of dimensions. Adaptive decisions are allowed at any router along the routing path. If a selected routing *way* has the least dimension among the other adaptive ways (i.e., it matches the one produced by DOR), then a buffer from *any* class can be allocated. Otherwise, a buffer from the *adaptive* class should be allocated. This algorithm will be referred to as ADAPTIVE and can be shown to be deadlock-free.

Finally, a fully adaptive deadlock-free minimal routing algorithm for *k*-ary *m*-way tori networks can be designed based on the DOR_RING algorithm as follows: A third buffer class, *adaptive*, is required in addition to the *low* and *high* classes used by the ring algorithm. Adaptive decisions are allowed at any router along the routing path. If a selected routing *way* matches the one produced by DOR and the RING algorithm matches the same group for the next buffer and destination processor, then the next buffer can be allocated from *any* class. If the selected routing *way* matches the one produced by DOR but the RING algorithm does not match the same group, then the next buffer can be allocated either from the *low* or from the *adaptive* class. If a selected routing *way* does not match the one produced by DOR then the next buffer should be allocated from the *adaptive* class only. This algorithm will be referred to as ADAPTIVE_RING and can be shown also to be deadlock-free.

4 Network Simulation and Results

To measure the performance of interconnection networks with multiway channels, I have simulated a mesh, a torus, and a hypercube *k*-ary *m*-way network varying few parameters in every run. The simulator is a C++ program that simulates *k*-ary *m*-way networks at the flit level. A flit transfer between two adjacent routers, over an *m*-way channel, is assumed to take place in one clock cycle. The network is simulated synchronously, moving all flits that have been granted channels in one clock cycle and then advancing time to the next cycle. The simulator can be configured to support different network sizes, dimensionalities, processor factors, buffers in a buffer set, buffer sizes, routing algorithms, arbitration algorithms, messages lengths, message generation rates, and traffic patterns. Flags indicating the use of full and availability status by a router can also be set. The simulator can generate various statistics, such as average message latency, maximum latency, latency standard deviation, latency histogram, channel utilization rate, node injection rate, and node ejection rate.

Latency is measured from the time a message is generated at a source node until the tail flit is ejected at a destination node. Source queuing time is included in the latency measurement. *Traffic* is measured as the percentage of utilization of channels. A channel is utilized during a clock cycle if it is used to transfer a flit successfully. The injection rate of a node is the percentage of channel cycles used to inject a flit successfully into the network. The ejection rate is the percentage of channel cycles used to eject a flit successfully from the network. The average traffic, injection, and ejection rates are taken over all channels and nodes in the network and over a period of time.

4.1 Effect of Increasing the Number of Buffers

The purpose of this experiment is to measure the effect of increasing the number of buffers in a buffer set. A medium size 3D-torus network with $8 \times 8 \times 8$ 7-way channels and 512 processor nodes is simulated. The DOR_RING algorithm is used. The traffic is uniform. All messages carry 64 bytes of data. They occupy 4 data flits + a header flit. Each flit is 16 bytes long and is transferable over a

channel in one clock cycle. The number of buffers in each set is 2, 4, and 8 respectively. However, the size of each buffer is fixed at 2 flits. The results of these experiments are shown in Figures 8 and 9, respectively.

The graphs of Figures 8 and 9 are not functions. The average latency, ejection rate, traffic, and latency standard deviation are all measured values. The independent parameter, specified to the network simulator, is the average period between message generations. The period between two message generations is a random value generated according to an exponential distribution. In other words, the message arrival rate is a Poisson distribution.

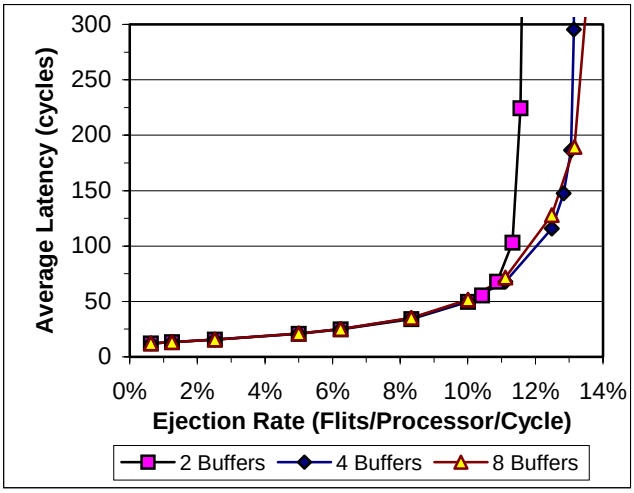

Figure 8: Average latency and Ejection Rate in an 8x8x8 Torus

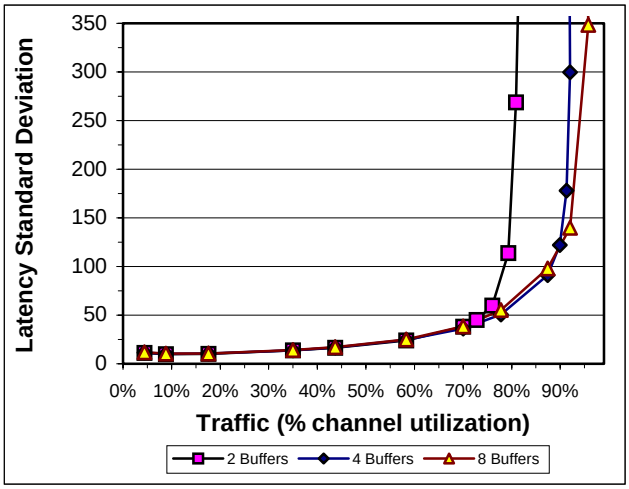

Figure 9: Latency Standard Deviation and Traffic in an 8x8x8 Torus

When the traffic is below saturation, the message latency is affected only slightly by the traffic. However, as the network saturates, latency starts increasing sharply. The latency standard deviation also varies with the traffic. Below saturation point, the latency standard deviation is a small value and increases only slightly with the traffic. However at saturation, the latency standard deviation increases also sharply. Saturation occurs when the nodes of a network generate messages at a higher rate than the one that can be handled by a network. These messages end up waiting at the source node queues. Increasing the number of buffers in each buffer set from 2 to 4 improved the ejection rate and the traffic. However, increasing it from 4 to 8 is not justifiable in this case.

4.2 Effect of Topology and Routing Algorithm

In this experiment an 8x8x8 mesh, an 8x8x8 torus, and a 9D hypercube network are simulated. All networks have 512 processor nodes. All buffer sets consist of 4 buffers each of size 2 flits. All messages carry 64 bytes of data (1 header + 4 data flits). The traffic pattern is uniform. The DOR and ADAPTIVE routing algorithms are used in the mesh and hypercube networks. The DOR_RING and ADAPTIVE_RING algorithms are used in the torus network. The results are shown in Figures 10 and 11.

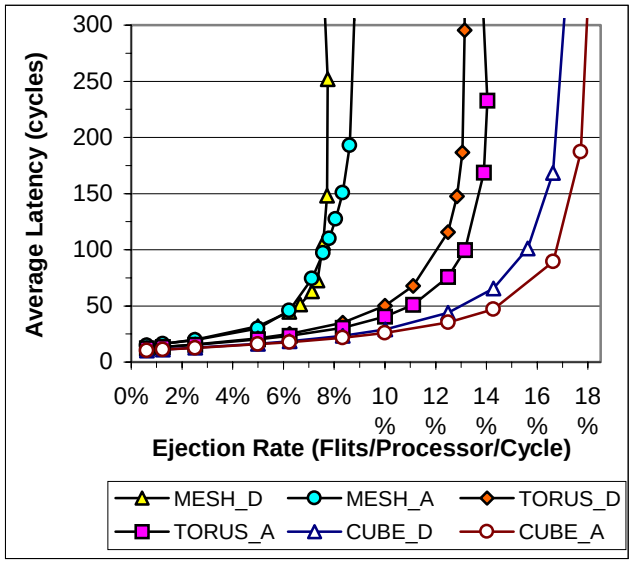

Figure 10: Average latency and Ejection Rate in a 3D Mesh, a 3D Torus, and a 9D Hypercube

The graphs of Figures 10 and 11 clearly indicate that the performance of a hypercube is better than that of a torus, which in turn is better than the performance of a mesh. A hypercube can provide the highest throughput (ejection rate) and lowest latency amongst all topologies. The reason is that the number of wires per channel is kept constant in a *k*-ary *m*-way network irrespective of the network topology or dimension and the distances between processors are the shortest in the case of a hypercube. However, there are other factors that can make the hypercube less attractive. For instance the sharing factor, *m*, of an *m*-way channel is 10 in the case of a 9D hypercube, while it is 7 in the case of a 3D mesh or torus with a processor factor of 1. Increasing

the dimension of a network will increase the costs of packaging, the lengths of wires, and reduces the speed of a channel. The clock period was assumed to be same in all network topologies, while it may increase as the network dimension increases. The board-level packaging of a *k*-ary *m*-way network is also an open problem that needs further study. Lower-dimensional networks are favored over higher-dimensional networks from the engineering point of view.

The performance of a torus is clearly better than the performance of a mesh. The throughput is almost twice as much. The traffic in a torus can easily exceed 95%, but it barely reaches 75% under adaptive routing in a mesh. The reason is that the traffic distribution is not uniform in a mesh topology even when the traffic pattern itself is uniform. The traffic is very heavy (near 100%) at the center of mesh at saturation, but is very light (about 20%) at the corners and boundary channels. This problem does not exist in a torus because it has a symmetric topology.

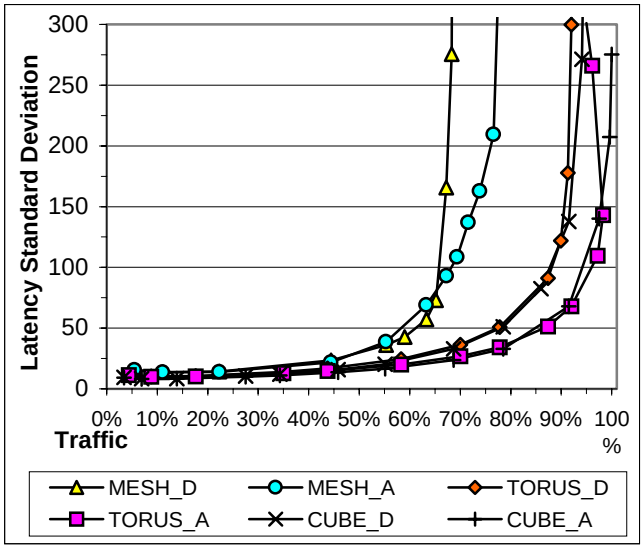

Figure 11: Latency Standard Deviation and Traffic in a 3D Mesh, a 3D Torus, and a 9D Hypercube

The results of Figures 10 and 11 also indicate that the ADAPTIVE routing algorithm, used in MESH_A and CUBE_A, and the ADAPTIVE_RING algorithm, used in CUBE_A, perform always better than deterministic routing algorithms. The performance improvement can be even more substantial when the traffic is not uniform.

5 Conclusion and Further Research

This paper presented a new class of interconnection networks called *k*-ary *m*-way networks. These networks are based on *m*-way channels and routers. The idea is to reduce the number of links per router to only two and to make channels very wide. The performance of *k*-ary *m*-way mesh, torus, and hypercube networks was evaluated under different routing algorithms. The initial results are encouraging and stimulate more research in this direction. The router discussed in this paper is described in VHDL. It is currently being extended to support broadcasting and multicasting. Further research in this direction is to link processor nodes within routers and to support routing in faulty networks.

References

[1] J.-C. Bermond and F. Ergincan, Bus Interconnection Networks, *Discrete Applied Mathematics*, 68, pages 1-15, 1996.

[2] L. Bhuyan and D. P. Agrawal, Generalized Hypercube and Hyperbus Structures for a Computer Network, *IEEE Transactions on Computers*, vol. 33, no. 4, pages 323-333, April 1984.

[3] J. Carbonaro and F. Verhoorn, Cavallino: The teraflops router and NIC, *Proceedings of Hot Interconnects Symposium IV*, August 1996.

[4] W. Dally and C. Seitz, Deadlock free message routing in multiprocessor interconnection networks, *IEEE Transactions on Computers*, vol C-36, no. 5, pages 547-553, May 1987.

[5] J. Duato and P. Lopez, Performance Evaluation of Adaptive Routing Algorithms in k-ary n-cubes, in *Proceedings of the Parallel Computer Routing and Communication Workshop*, May 1994.

[6] J. Duato, A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks, *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pages 1055-1067, October 1995.

[7] A. Ferreira, A. G. vel Lejbman, and S.W. Song, Broadcasting in bus interconnection networks, *CONPAR 94*, Lecture Notes in Computer Science, Springer-Verlag, September 1994.

[8] E. Fleury, P. Fraigniaud, A General Theory for Deadlock Avoidance in Wormhole-Routed Networks, *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 7, pages 626- 638, July 1998.

[9] M. Mudawwar, A Switch-Free Router for *k*-ary *m*-way Networks, *submitted to the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2000.

[10] M. Noakes, D. Wallach, and W. Dally, The J-Machine Multicomputer: An architecture evaluation, *Proceedings of the 20th International Symposium on Computer Architecture*, pages 224-235, May 1993.

[11] S. L. Scott and G. Thorson, The Cray T3E network: adaptive routing in a high performance 3D torus, *Proceedings of Hot Interconnects Symposium IV*, August 1996.

[12] T. Szymanski, Hypermeshes: Optical interconnection networks for parallel processing, *Journal of Parallel and Distributed Computing*, vol. 26, pages 1-23, January 1995.