In Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications

A Switch-Free Router for *k*-ary *m*-way Networks

Muhammed F. Mudawwar

Computer Science Department
The American University in Cairo
113 Kasr el Aini street, Cairo, Egypt
mudawwar@aucegypt.edu
Tel: +20-2 3575305
Fax: +20-2 3557565

Abstract

k-ary m-way networks are multi-dimensional mesh and tori networks based on m-way channels. An m-way channel is the physical wiring of m links. An m-way router interfaces two m-way channels only, irrespective of the network topology or dimension. This has important advantages: the same router can be used to build networks of various dimensionalities and topologies; physical channels can have very wide data links; and broadcasting and multicasting are facilitated. The design of a switch-free m-way channel wormhole router is detailed in this paper. Channel arbitration, buffer management, and routing are presented. The performance of k-ary m-way networks is evaluated.

Keywords: *m*-way channel, *k*-ary *m*-way network, *m*-way router architecture, performance evaluation.

1 Introduction

Direct interconnection networks use direct links between routers. 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 [2] [3] [8] [9] [10].

In this study, I am proposing a new network topology, called *k*-ary *m*-way network, which is based on the concept of *m*-way channels. The idea of an *m*-way channel is that a maximum number, *m*, of routers and processors can link directly to it, and hence share the same physical channel. *k*-ary *m*-way networks can be viewed as the dual of *k*-ary *n*-cube networks. Routers, called *m*-way routers, are designed to interface two *m*-way channels only, irrespective of the network topology or dimension. This is not possible with traditional wormhole routers. *k*-ary *m*-way networks are detailed next.

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

An *m*-way channel (called also *multiway channel*) is a physical channel *shared* by a fixed number, *m*, of routers and nodes. It is the physical wiring of *m* router/processor links. An *m*-way router interfaces two *m*-way channels only. It defines the operation of an *m*-way channel. At any clock cycle, only one of the *m* routers or processors can drive the channel. However, all *m* routers and nodes concurrently read the channel. *m*-way channels and routers can be used to build a variety of network topologies [7].

A *k*-ary *m*-way network is a multi-dimensional mesh or torus constructed using *m*-way 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.

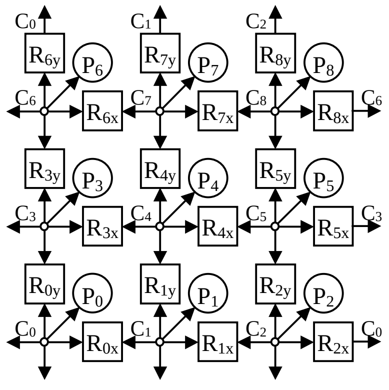


Figure 1: 3-ary 5-way Torus

An example of a 3-ary 5-way torus is shown in Figure 1. This is a 2-dimensional torus with 9 processors and 18 routers. Each 5-way channel wires 4 routers directly to a processor. 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*.

A k-ary $(2n + p)$-way network (where $m = 2n + p$) is constructed 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. p processor nodes are then linked directly to each channel. A k-ary $(2n + p)$-way network has k^n m-way channels, $p\ k^n$ processors, and $n\ k^n$ routers in case of a torus, or $n\ (k-1)\ k^{n-1}$ routers in case of a mesh network. A channel address is an n digit, radix k number: $c = a_{n-1}\ldots a_1 a_0$. Each address digit a_i represents a channel's coordinate in dimension i 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 (mesh) or by $\pm$ 1 mod k (torus). A processor linked to channel $a_{n-1}\ldots a_1 a_0$ has address $a_{n-1}\ldots a_1 a_0, l$ where l is the local processor address and can assume the values 0 through $p-1$. If $p = 1$ then a node address becomes equal to its corresponding channel address. There are $2n$ routers linked to each channel. 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.

m-way routers interface two channels, regardless of the network topology or dimension. This is a distinguishing feature of multiway channel networks, of which k-ary m-way networks are a special case. This has important advantages. First, an m-way channel can be made much wider than a direct channel. Given a packaging technology, say about 300 I/O pins per router chip, an m-way router can be designed to interface channels that are 128-bit data wide. The 8-way router design that will be discussed in this paper defines 8-way channels with 128 data and 17 control lines. A total of 290 I/O pins are used for both channel interfaces. Contrast this 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, each channel in a k-ary m-way network can be made much wider. The overhead of control lines is also less in a k-ary m-way network.

A second advantage of using m-way channels and routers is that the same router chip, if carefully designed, can be used to implement different networks. For instance, an m-way router can be used to implement low-dimensional and high-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 received in multiple routers concurrently during the same clock cycle.

3 Router Architecture

The internal structure of an m-way channel router is depicted in Figure 2. A router has two channel interfaces, two channel arbitrators, and two sets of buffers with allocation/mapping units, routing logic, and buffer arbitrators. The dimensionality of a router, *DIM*, is specified through external pins. The *directionalities* of the two sets of buffers are DIM+ and DIM-. The directionality is used to identify a buffer set when selecting a driver for a channel or when transferring message flits. This identification should be unique across an m-way channel. In the case of a processor node, we need also to uniquely identify its injection and ejection buffer sets. Observe that no crossbar switch is required. This simplifies the implementation of a router and makes it faster.

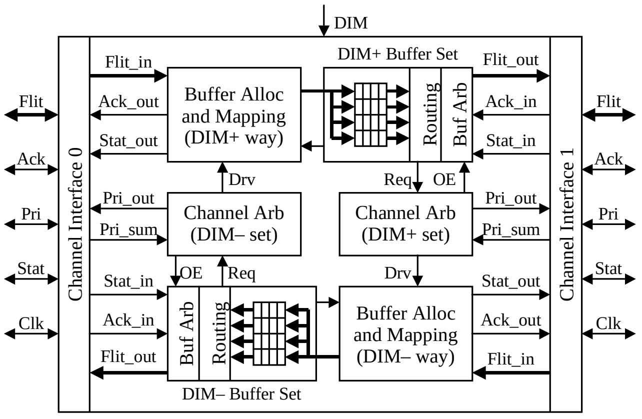


Figure 2: Internal Structure of a Router

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 sum of output priorities of requesting drivers. The *Stat* lines carry the availability and full status of receiver buffers in a cyclic fashion. The *Clk* line is used to synchronize the operation of an m-way channel.

A buffer set can concurrently receive a flit while transferring a second one across a channel. To avoid the use of a global clock, each channel is designed to operate on a separate clock. A buffer set operates on two clocks because it interfaces two channels. The first clock synchronizes the receiving of a flit, while the second one synchronizes the transfer of a flit. An arbitrary skew can exist between the two clocks.

3.1 Message Format

A message consists of a header flit followed by an arbitrary number (possibly zero) of body flits, followed by a tail flit. A *Tag*, identifying the kind of flit, is generated at the sending node and transmitted with each flit. Four tags are used: *Header* (*H*), *Body* (*B*), *Tail* (*T*), and *Invalid* (*I*). This is depicted in Figure 3. A valid tag is a request to transfer a flit across an *m*-way channel. In addition to the tag, the driver's buffer number, *Buf*, is transmitted with every flit. This number identifies the driver's buffer and can change on every router a message reaches. The tag and driver's buffer number are control information sent with every flit.

H	Buf	Way	Class	Dest	Len	Other Control
B	Buf	Data				
B	Buf	Data				
B	Buf	. . .				
T	Buf	Data				

Figure 3: Message Format

The header flit carries additional control information. It carries the routing *Way*, which specifies the direction of a header flit when it is transmitted across an *m*-way channel. It is a number between 0 and *m*-1. This field is defined at each router a header flit reaches. If broadcasting or multicasting on an *m*-way channel is to be supported then *Way* becomes an *m*-bit field, where each bit is associated with one direction. The routing *Way* is required to transfer header flits, but is not required to transfer body or tail flits. This is why it is encoded in the header flit only. The *Class* field identifies the buffer class or the subset of buffers that can be allocated at the receiver side. An adaptive routing algorithm may divide buffers into classes (possibly non-disjoint) to restrict allocation and to avoid deadlocks. Such buffer class information should be carried by a header flit to guide buffer allocation at the receiver side. The destination processor address, *Dest*, is another header field that can be encoded either absolutely or relative to the source processor address. If a router is designed for networks of different dimensionalities then the destination field should have different interpretations of sub-fields. The length field, *Len*, encodes the number of data bytes in the flits that follow the header flit. The length field can be used to allocate storage at the destination node as soon as a header flit is received. It can be also used to ensure the correctness of the tags and to establish a limit on the length of a message.

Additional control information can be carried by a header flit. For instance, a non-minimal routing algorithm may require the number of misrouting steps to be carried in the header flit to avoid livelocks. Other control information can also include the priority of a message or the address and context of a thread to be executed at the destination node when a message header flit is received.

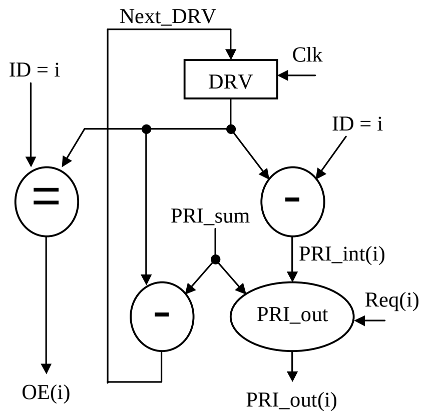
3.2 Channel Arbitration

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 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. Thus, it cannot assign fixed priorities to drivers.

At the beginning of each clock cycle, *m* requests, *Req*, are generated internally by the *m* drivers (buffer sets) of a given channel (see Figure 2). These *m* requests are input to all the channel arbitrators. The channel arbitrators concurrently compute their internal priorities. The internal priority is unique in every arbitrator. Output priorities, *Pri_out*, are produced at the outputs of all arbitrators. The output priorities are wired together on the channel priority lines, *Pri*, to produce the wired-OR sum. The requester with the highest internal priority wins and its priority appears on the channel priority lines. All channel arbitrators read the channel priority lines and update an internal register that holds the current channel driver. This is done concurrently with the edge of a clock that synchronizes the operation of all drivers interfacing a channel.

The channel arbitrator can generate the internal priorities according to Round Robin. This implementation is shown in Figure 4. A maximum of 8 drivers, uniquely identified as 0 to 7, are allowed across a channel. The internal priority of driver *i*, *PRI_int*(*i*), is computed as *Drv* – *i*, where *Drv* is the current driver number in all channel arbitrators and *i* is the driver's identifier (DIM+ or DIM–). The subtraction is done modulo 8. Because *i* is unique in every driver, the internal priorities are guaranteed to be unique. The internal priorities also change in cycles (round-robin) when the current driver changes. The output priority of driver *i*, *PRI_out*(*i*), is a function of its internal priority, *PRI_int*(*i*), the sum priority, *PRI_sum*, and the request input, *Req*(*i*), from the buffer set. The sum priority, *PRI_sum* = $s_2s_1s_0$, is a 3-bit vector that represents the wired-OR sum of all output priorities. The sum priority is fed as input to all channel arbitrators to adjust their output priorities. The output priorities are adjusted according to the table of Figure 4. *PRI_out*(*i*) is defined such that the sum priority, *PRI_sum*, will be equal to the highest internal priority of a requesting driver.

Once the sum priority is determined, it is subtracted from the current driver number, *DRV*, to determine the next one, *Next_DRV*. The driver register that holds the current driver number is updated at the beginning of a clock cycle. The update is done concurrently in all channel arbitrators. The current driver number is used to enable the output of exactly one driver ($OE(i)$ = 1 when DRV = i). Channel arbitration does not waste clock cycles. It is done concurrently while transferring a message flit.



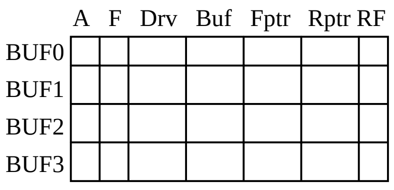
$$PRI\_sum = s_2 s_1 s_0 = \sum_{i=0}^{7} PRI\_out(i)$$

PRI_int(i)	PRI_out(i)		
000	0	0	0
001	0	0	$s_2'.s_1'$.req(i)
010	0	s_2'.req(i)	0
011	0	s_2'.req(i)	s_2'.req(i)
100	req(i)	0	0
101	req(i)	0	s_1'.req(i)
110	req(i)	req(i)	0
111	req(i)	req(i)	req(i)

Figure 4: Channel Arbitrator and Output Priority Function

3.3 Buffer State

Each buffer in a buffer set has associated state information, as depicted in Figure 5. The allocation bit, *A*, indicates the allocation status. The full bit, *F*, indicates the full status. The driver number, *Drv*, indicates the driver set from which the flits of a message are received. The driver's buffer number, *Buf*, specifies a buffer in a buffers set. *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.



A
F
Drv
Buf
Fptr
Rptr
RF
BUF0
BUF1
BUF2
BUF3

A:	Allocation bit
F:	Full bit
Drv:	Driver number
Buf:	Driver's Buffer number
Fptr:	Front Pointer
Rptr:	Rear Pointer
RF:	Receiver's Full status

Figure 5: Buffer associated state information

3.4 Buffer Allocation

An *m*-way channel operates as follows: 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 receive the header flit. However, only one buffer allocation unit will accept the header flit, depending on the routing *Way*. In the case of multicasting and broadcasting, it is also possible to have several buffer allocation units concurrently accepting the header flit. Once accepted, the buffer allocation unit will allocate a buffer for the header flit and send back an acknowledgment, *Ack_out*.

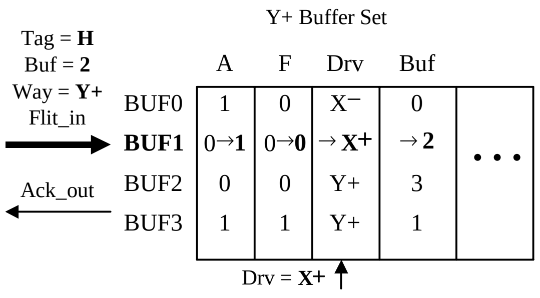


Figure 6: Allocation of a buffer in a buffer set

Figure 6 illustrates the allocation of a buffer to a header flit. All channel arbitration units at a given channel have concurrently selected buffer set X+ to be the current driver. Observe that the current driver, *Drv*, is passed from the channel arbitrator to the buffer allocation unit as shown in Figure 2. The header flit carries the driver's buffer number, which is 2 in Figure 6. The driver's number, *Drv*, and the driver's buffer number, *Buf*, are stored in the allocated entry. This allows each buffer to identify the *previous* buffer along the routing path.

3.5 Buffer Mapping

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 receive the flit that carries the driver's buffer number, *Buf*. They also receive the current driver number, *Drv*, from the channel arbitrator. All allocation and mapping units are searched in parallel 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 it. An acknowledgment is sent back.

An allocated buffer can be freed as soon as a tail flit is received. This is called *early buffer freeing*. Alternatively, we can free a buffer after a tail flit is transmitted, which is called *late buffer freeing*. With early buffer freeing, it is possible to have more than one message stored in a buffer.

This is not a problem because the tags, which are stored with the data of each flit, indicate the boundaries of each message. Early buffer freeing improves the utilization of buffers. However, the analysis of deadlock-free adaptive routing algorithms with early buffer freeing becomes more complex.

Allocating buffers and mapping them at the receiver side rather than on the driver side has several advantages. It reduces the number of control lines because the routing *way* is carried only in the header flit and is not required in the body and tail flits. It also simplifies the implementation because a driver does not have to keep track of the allocation status of every buffer across an *m*-way channel. Furthermore, broadcasting and multicasting are facilitated because only the driver's buffer number has to be carried in every flit, rather than the numbers of all allocated buffers at the receiver sides.

3.6 Routing

Routing across an *m*-way channel is to determine the next router and buffer along the routing path. A routing algorithm is defined as a routing function and a selection function. It is implemented in the routing logic at the driver's side as well as in the allocation unit at the receiver's side. The routing logic at the driver's side determines the routing *Way* and buffer *Class* for a header flit, denoted as (*Way*, *Class*). The routing *Way* specifies the next router across an *m*-way channel. The buffer *Class* specifies a subset of buffers. It is used to ensure deadlock freedom for some adaptive 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. The allocation unit at the receiver side accepts a header flit only if it along its *Way* and allocates a new buffer that belongs to the specified buffer *Class*. The same routing algorithms and deadlock-avoidance theories discussed in the literature [4], [6] for wormhole-routed *k*-ary *n*-cube networks are also applicable to *k*-ary *m*-way networks.

4 Network Simulation and Results

To measure the performance of interconnection networks with multiway channels, I have simulated a number of mesh networks 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, takes 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 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, injection rate, and ejection rate.

Latency is measured from the time a message is queued at a source processor until the tail flit is ejected at a destination processor. 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/ejection rate of a processor is the percentage of channel cycles used to inject/eject a flit successfully into/from the network. The injection and ejection rates are equal at steady state.

4.1 Effect of Increasing the Buffers and their Sizes

The purpose of this experiment is to measure the effect of increasing the buffer size and the number of buffers in a buffer set. A medium size 2D-mesh network with 16×16 5-way channels and 256 nodes is simulated. Dimension-order routing 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 first experiment uses 1 buffer only in each set. However, the size of the buffer is varied from 1 to 128 flits. The second experiment varies the number of buffers in each set from 1 to 16. However, the size of each buffer is fixed at 2 flits. Each simulation run took 100,000 cycles of which the first 30,000 cycles are startup. Startup cycles are ignored in the statistics. The results of these experiments are shown in Figures 7 and 8, respectively.

The graphs shown below are not functions. Both the latency and the traffic are measured values. When the traffic is below saturation, the message latency is affected only slightly by the traffic. However, as the network saturates, latencies start increasing sharply. The latency standard deviation, not shown in the figures, also varies with the offered load and traffic. Below saturation point, the latency standard deviation is a small value that is almost a constant. However, the latency standard deviation increases sharply beyond the saturation point. Saturation occurs when the nodes of a network generate messages at a higher rather than they can be delivered. These messages end up waiting at the source node queues. Adding more buffers or increasing the depth of buffers improves the injection and ejection rates and allows the network to accept more traffic. Increasing the depth of a buffer from 1 to 8 flits in Figure 7 improves the traffic significantly. However, little improvement is obtained beyond this buffer size. Similarly, increasing the number of buffers in a set from 1 to 4 is justifiable. Adding more buffers increases the cost and latency of hardware and does not provide significant improvement.

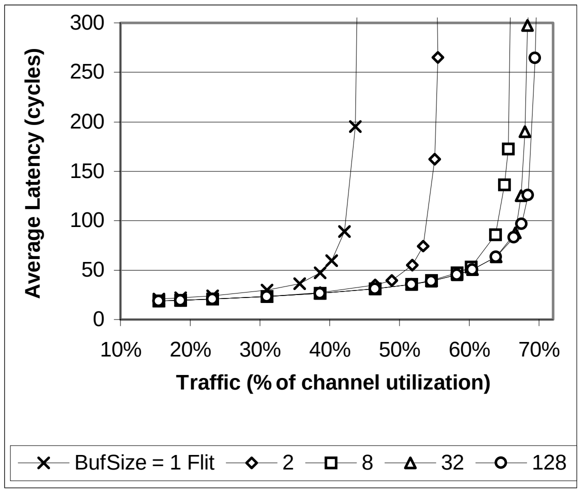


Figure 7: Effect of increasing the buffer size

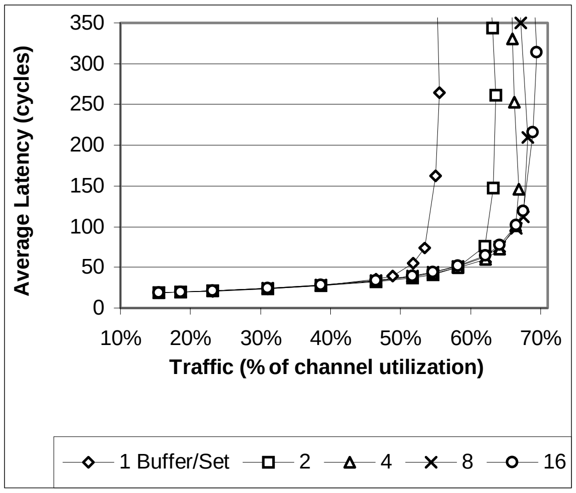


Figure 8: Effect of increasing the number of buffers

5.2 Traffic Distribution in a Mesh

The traffic of Figures 7 and 8 did not exceed 70%. The question is why? To answer this question, it is important to examine the traffic distribution in a mesh. The traffic distribution or channel utilization is not uniform, even when the generated traffic is uniform. The traffic distribution is shown in Figure 9. The traffic is saturated (about 100%) at the center of the mesh, while it is very light (about 20%) at the corners. This is a property of the mesh topology because it is not a symmetric network. The problem can be alleviated using a torus network, but with added cost. It can also be alleviated if the generated traffic exploits locality.

5.3 Effect of Network Dimension, and Processor Factor

In this experiment eleven networks are simulated. All networks have 512 processors. 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. Dimension order routing is used. The results are shown in Figure 10. The first graph shows 2D and 3D networks. The second graph shows 4D, 7D, 8D, and 9D networks. The processor factor varies from 1 to 4. For example, the network 32×16×P1 is a 2D mesh with 32 channels along the X dimension, 16 channels along the Y dimension, and a processor factor of 1. The network 8×4×4×P4 is a 3D mesh with 8 channels along the X dimension, 4 channels along the Y and Z dimensions, and a processor factor of 4. The network 8D-HC×P2 is an 8D-hypercube network with a processor factor of 2. The total number of channels varies from 128 to 512 depending on the processor factor. The size, cost, and bandwidth of a network decrease when the processor factor increases.

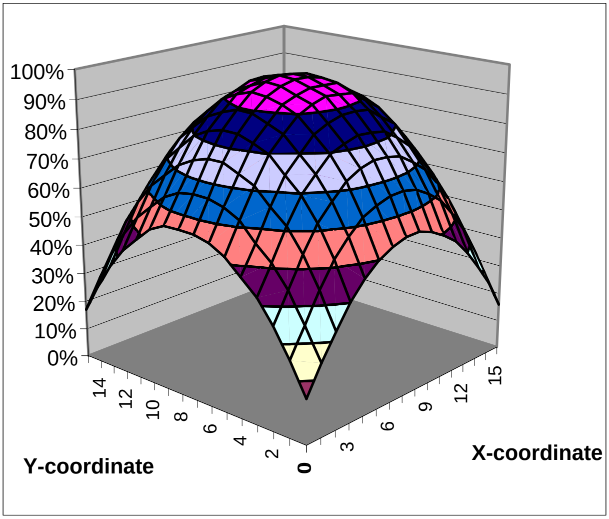
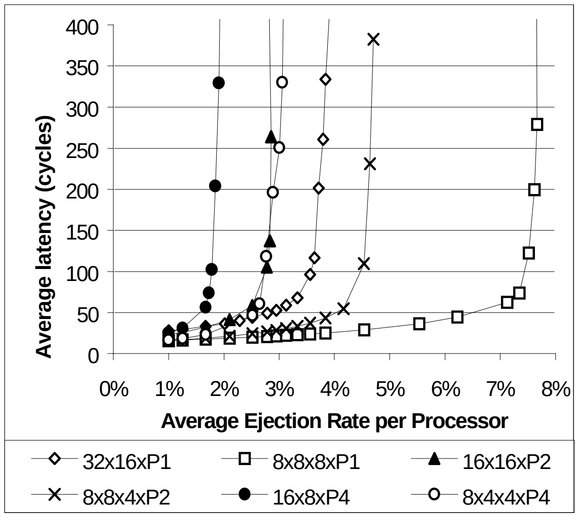


Figure 9: Traffic Distribution in a 16 × 16 mesh network

The results of Figure 10 reveal that high-dimensional *k*-ary *m*-way networks perform better than low-dimensional ones. However, they have a higher cost. The higher the ejection rate, the more the number of flits (or messages) that can be transferred in a network in a given period of time. The highest performer is the 9D-HC×P1 hypercube network, which also has the highest number of routers and cost. The average ejection rate per processor exceeds 17%. The network traffic reaches 95% at saturation point (not shown in Figure 10). At steady state, the injection rate is equal to the ejection rate. Thus, 17% + 17% = 34% of channel cycles is used for injection and ejection. The remaining 95% – 34% = 61% is used to transfer flits between routers. The lowest performer is the 2D 16×8×P4 network, which also has the least number of routers. The average injection rate per router is about 1.9% and the network traffic is 68% at saturation. Because 4 processors are wired to a channel in the 16×8×P4 network, 4 × 1.9% × 2 = 15.2% of channel cycles are used for injection and ejection. This leaves 68% – 15.2% = 52.8% of channel cycles for transferring flits between routers.



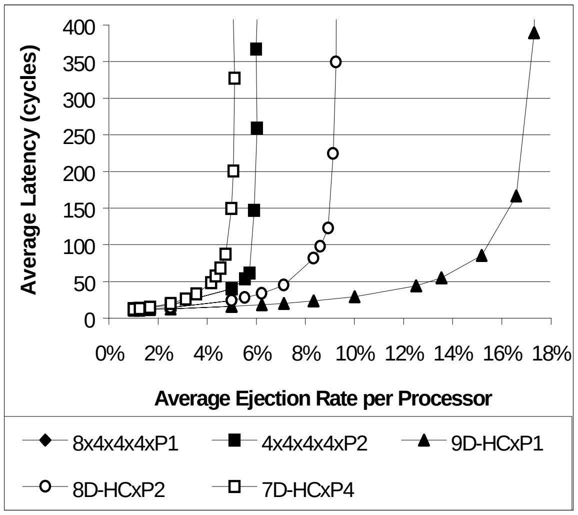


Figure 10: Effect of network size, dimension, and node factor in low and high dimensional networks

It is interesting to observe that the 7D-HC×P4 hypercube network with 448 routers outperforms the 3D 8×8×4×P2 network with 640 routers, which also outperforms the 2D 32×16×P1 network with 976 routers. The average injection rate at saturation point is 5.1% for the 7D-HC×P4 hypercube network, while it is 4.7% for the 3D 8×8×4×P2 network and 3.9% for the 2D 32×16×P1 network. Similarly, the 8D-HC×P2 hypercube network with 1024 routers outperforms the 3D 8×8×8×P1 network with 1344 routers and comes close to the 4D 8×4×4×4×P1 with 1600 routers.

6 Conclusion and Further Research

This paper introduced a new class of interconnection networks called *k*-ary *m*-way networks. These networks are based on *m*-way channels. The idea is to reduce the number of links per router to only two and to make channels very wide. The design of an *m*-way switch-free router was detailed in this paper. The performance of multi-dimensional mesh and hypercube networks was evaluated. The initial results are encouraging and stimulate more research in this direction. The router discussed in this paper is described in VHDL and was tested for correctness. It is currently being extended to support broadcasting and multicasting. Further research in this direction is to study the engineering-aspects of system packaging for higher dimensional networks and to support fault-tolerant routing.

References

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

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

[3] Dally W., et al., Architecture and implementation of the Reliable Router, *Proceedings of Hot Interconnects Symposium II*, August 1994.

[4] 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.

[5] Duato J., Yalamanchili S., and Ni L., *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, 1997.

[6] 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.

[7] Mudawwar M., Multiway Channels in Interconnection Networks, in *Proceedings of the ISCA 12th International Conference on Parallel and Distributed Computing Systems*, August 1999.

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

[9] W. Oed, The Cray Research Massively Parallel Processing System: Cray T3D, Cray Research, 1993.

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

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