Multiway Channels in Interconnection Networks*

Muhammed F. Mudawwar

Computer Science Department
The American University in Cairo
mudawwar@aucegypt.edu

Abstract

This article introduces the concept of a *multiway channel*, a physical channel shared by a fixed number of routers and nodes. A router designed around the concept of a multiway channel interfaces two channels only, irrespective of the network topology. This has two immediate advantages: The same router can be used to build networks with different dimensionalities or topologies and physical channels can be very wide (typically 128-bit-data wide). This is to be contrasted with state of the art wormhole routers, designed for 2D and 3D networks, which typically have 8-bit or 16-bit data links. The design of a new wormhole router, based on the concept of multiway channel, is detailed in this paper. The performance of networks, that use multiway channels, is evaluated. The results are very encouraging and promise a new line of research.

Keywords: multiway channel, wormhole router, interconnection networks, router architecture.

1. Interconnection Networks

The interconnection network is often the critical part of a massively parallel computer because application performance is very sensitive to network latency and throughput. An interconnection network is characterized by its topology, routing, and flow control. The topology is the arrangements of nodes and channels into a graph. Routing specifies how a message chooses a path in this graph. Flow control deals with the allocation of channel and buffer resources to a packet as it traverses this path [Dally 92].

Topologies that have been widely used in recent parallel computers include two- and three-dimensional meshes and tori. The 2D and 3D interconnection topologies can be found in Intel Paragon [Paragon 91], Stanford DASH [Lenoski 92], Stanford FLASH [Kushkin 94], MIT Alewife [Agarwal 90], MIT J-Machine [Noakes 93], MIT Reliable Router [Dally 94], Cray T3D [Oed 93] and Cray T3E [ScottThorson 96]. Other popular topologies that are less commonly used are the hypercube and the fat tree.

The architecture of a generic wormhole router [Duato 97] is shown in Figure 1. It includes input and/or output *buffers* for storing flits (flow control units). Each input buffer can also be subdivided into several virtual channel buffers. A crossbar *switch* is used for connecting input buffers to output buffers. A *routing and arbitration unit* implements the routing algorithm and selects the output link for an input message. If multiple messages simultaneously request the same output link, this component must provide for arbitration between them. *Link controllers* (LC) are used to control the flow of message flits across a physical channel. The *processor injection and ejection channels* implement a physical channel interface from/to the local processor.

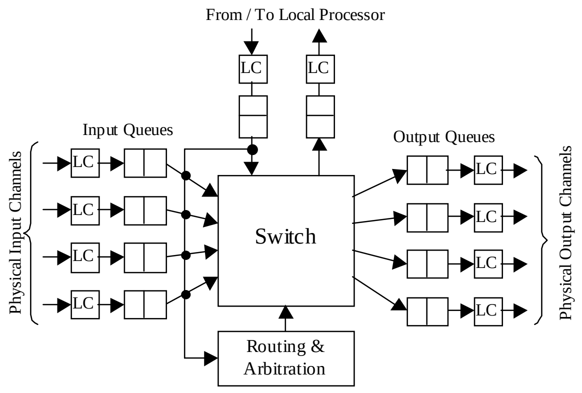


Figure 1: Generic wormhole router used in *k*-ary 2-cube networks

Router elements in the *k*-ary *n*-cube network have many dedicated input and output physical channels. For instance, routers used in 2D networks require four input and four output channels for communication with adjacent routers, as well as a fifth input and output channel for communication with the local processor. Routers used in 3D networks require six input and six output channels for communication with adjacent routers. As the dimensionality of the network increases, the number of input and output physical channels also increases. Since the number of I/O pins in router chips are limited by the packaging technology, the increase in the number of physical channels will decrease the number of wires and thus the bandwidth of a single physical channel. State of the art wormhole routers are designed for low dimensional networks (2D or 3D) with physical channels that typically are 8-bit-data or 16-bit-data wide.

In this study, I am proposing a new concept called a *multiway channel*. The idea of a multiway channel is that a fixed number of routers and nodes can share the same physical channel. Routers are designed to interface two physical channels only, irrespective of the topology of the

* This work is supported by a research grant from the American University in Cairo.

network. This is not possible with current wormhole routers. The concept of the multiway channel is detailed next.

2. The Multiway Channel Concept

A multiway channel is a physical channel *shared* by a fixed number of routers and nodes. Only one router or node can drive the physical channel at any given clock cycle. We call that router or node a *driver*. However, all routers and nodes connected to a multiway channel can concurrently monitor the channel. Thus, at a given clock cycle, exactly one of the routers or nodes acts as a driver but all of them act as monitors of the physical channel.

An example of a two-dimensional torus with 16 nodes, 32 routers, and 16 5-way channels is shown in Figure 2*a*. Each 5-way channel is a set of wires connecting 4 routers to a node. Each channel is identified by its *y* and *x* coordinates as *Cy,x*. A node connected to channel *Cy,x* is identified as *Ny,x*. Routers connected to channel *Cy,x* are identified as *Xy,x* and *Xy,x*-1 when they are along the *X* dimension, and *Yy,x* and *Yy*-1,*x* when they are along the *Y* dimension.

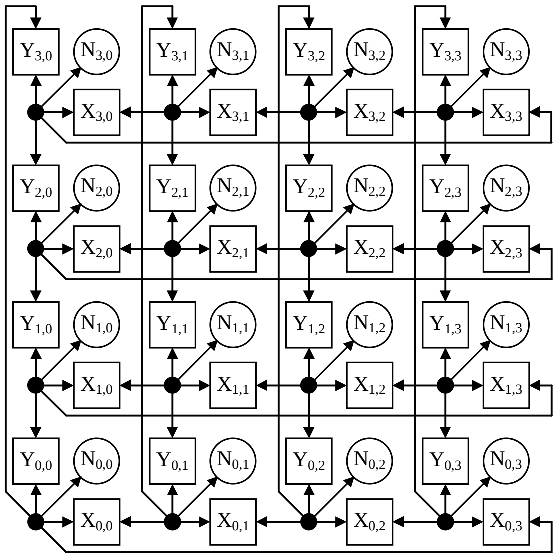

$Y_{3,0}$ $N_{3,0}$ $Y_{3,1}$ $N_{3,1}$ $Y_{3,2}$ $N_{3,2}$ $Y_{3,3}$ $N_{3,3}$
$X_{3,0}$ $X_{3,1}$ $X_{3,2}$ $X_{3,3}$
$Y_{2,0}$ $N_{2,0}$ $Y_{2,1}$ $N_{2,1}$ $Y_{2,2}$ $N_{2,2}$ $Y_{2,3}$ $N_{2,3}$
$X_{2,0}$ $X_{2,1}$ $X_{2,2}$ $X_{2,3}$
$Y_{1,0}$ $N_{1,0}$ $Y_{1,1}$ $N_{1,1}$ $Y_{1,2}$ $N_{1,2}$ $Y_{1,3}$ $N_{1,3}$
$X_{1,0}$ $X_{1,1}$ $X_{1,2}$ $X_{1,3}$
$Y_{0,0}$ $N_{0,0}$ $Y_{0,1}$ $N_{0,1}$ $Y_{0,2}$ $N_{0,2}$ $Y_{0,3}$ $N_{0,3}$
$X_{0,0}$ $X_{0,1}$ $X_{0,2}$ $X_{0,3}$

Figure 2a: 2D Torus with 16 nodes, 32 routers, and 16 5-way channels

The structure of a multiway channel is shown in Figure 2*b*. A channel consists of data and control lines. The control lines include the arbitration lines used for the selection of the channel driver for the next clock cycle. The same wires are connected to all router and node interfaces. Although only one node is shown connected to a channel, it is also possible to have several nodes.

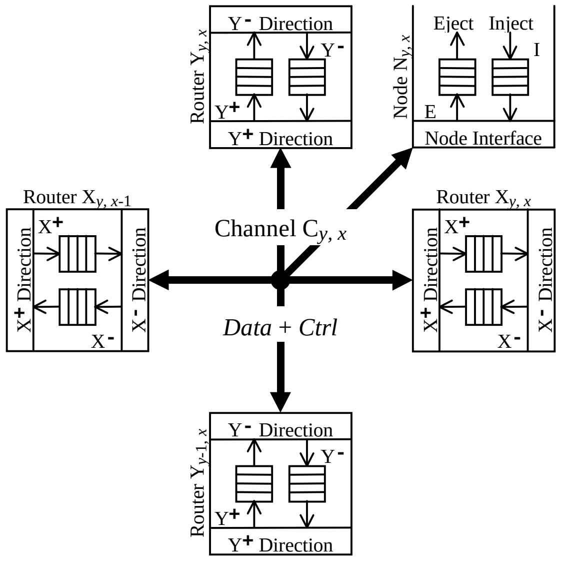


Figure 2b: Structure of a Multiway Channel

2.1 Modified Network Topologies

A new spectrum of network topologies can be obtained by exchanging the role of routers and channels in classical networks. Popular topologies include multi-dimensional meshes, tori, hypercubes, and trees. Examples of two modified network topologies using the multiway channel concept are shown in Figure 3.

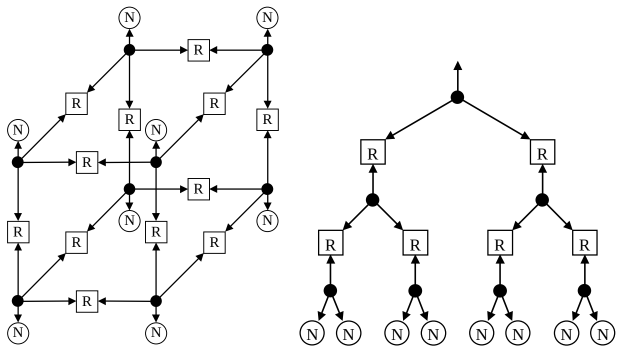


Figure 3: 3D cube with 4-way channels and a tree with 3-way channels

Routers based on the concept of multiway channels have two channel interfaces and are connected to two physical channels only, regardless of the network topology. An immediate advantage of this approach is that the width of a physical channel can be very wide. For a given packaging technology, say about 300 I/O pins, a router can be designed to interface physical channels that are 128-bit-data wide. This is 8 times the width of physical channels used in state of the art wormhole routers, such as the Intel Teraflops router [Carbonaro 96], the Cray T3E router [ScottThorson 96], and the MIT Reliable Router [Dally 94]. With extra-wide channels, a network can achieve higher throughput and lower message latency.

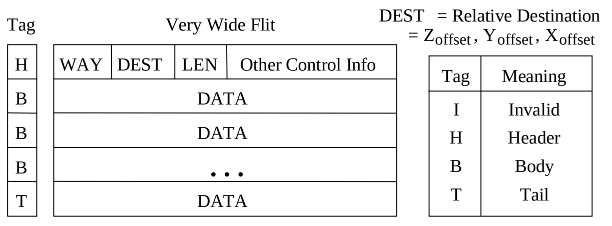
advantage of the multiway channel concept is that a router can be designed to operate under networks with different dimensionalities.

A second immediate advantage of using multiway channels is that the same router chip (if carefully designed) can be used to implement different topologies. This is because routers are connected to two physical channels only. For instance, the same router chip can be used to implement low-dimensional and high-dimensional networks. This is not possible with current wormhole routers.

2.2 Message Format

Message routing techniques such as *store-and-forward* and *virtual-cut-through* can be used in networks with multiway channels. However, I will restrict my discussion to the *wormhole* technique because of its low buffering requirement and good performance.

In a wormhole-routed network, a message is divided into flow control units, called *flits* [DallySeitz 87]. 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 4.



Tag	Meaning
I	Invalid
H	Header
B	Body
T	Tail

Figure 4: Message Format

The header flit carries the relative destination address, the routing way, and the length of a message. The relative destination address, *DEST*, encodes the destination offsets along the different dimensions. Different encodings can be used with different topologies. The routing way, *WAY*, is defined at each router a header flit reaches along its path from the source node to the destination node. The routing way is a number between 0 and *W*-1 that represents the direction that a router has chosen for a header flit, where *W* is the total number of ways for a multiway channel. The length field, *LEN*, encodes the number of data flits that follow the header flit. This number should be greater than zero because a message must have at least a tail flit. If *l* bits are used to encode the length of a message then the number of data flits that can follow the header is between 1 and 2^l (2^l is encoded as zero). The length field is redundant because the tag already identifies all the flits of a message. However, the length field can be used to allocate storage at the destination node as soon as the header flit is received. The length field 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 routing algorithm may need extra information to be transferred in the header flit, such as the *dimension reversal* number, DR, used in some routing algorithms [DallyAoki 93]. Other control information can also include the address and context of a thread to be executed at the destination node when a message header flit is received.

3 Router Architecture

The internal architecture of a router is depicted in Figure 5. A router has two channel interfaces identified as the positive and negative sides of a router. In addition, we need to identify whether a router is along the X dimension, Y dimension, etc. The dimensionality of a router, *DIM*, is hardwired through external pins. The dimensionality of a router is combined with the polarity of a channel interface to obtain a unique identification for a channel interface across a multiway channel. The channel interface identification defines also its directiona-lity. It is used for routing and for physical channel arbitration. The directionality of a channel interface means that the interface accepts flits routed in its direction. For instance, a channel interface identified as X+ can accept message flits going in the X+ direction. In the case of a processing node, we need also to uniquely identify its channel interface.

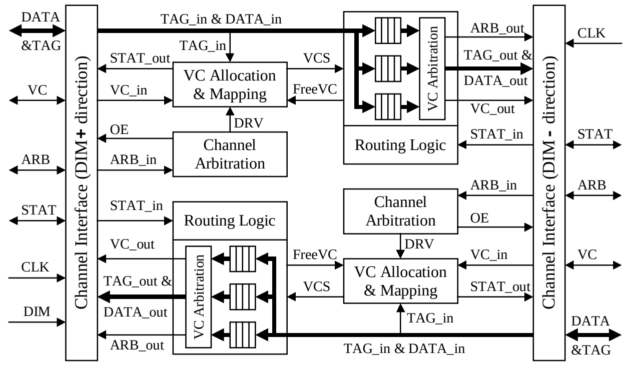


Figure 5: Internal Architecture of a Router

A physical channel consists of data, control, and arbitration lines. The *DATA* lines are used to carry one flit of a message at a time. The *TAG* lines are used to carry the tag of a flit. The *VC* lines are used to carry the Virtual Channel number of a flit. The *STAT* lines are used to carry *W* allocation status of the virtual channel buffers in the *W* routers and nodes attached to a multiway channel. The *ARB* lines are used to carry the *W* requests of the *W* channel drivers. A channel arbitrator uses the *ARB* lines to determine the next channel driver for the next clock cycle. Finally, the *CLK* input is used to synchronize the operation of a multiway channel. Each multiway channel has its own clock. This means that two different clocks drive the two sides of a router, as shown in Figure 5.

A router includes two sets of the following components: channel arbitrator, routing unit, virtual channel allocation and mapping unit, and virtual channel buffers. The channel arbitrator is used to determine the next channel driver. The routing logic implements the routing algorithm and determines the routing way of a header flit. The virtual channel buffers are used to buffer the flits of different messages. The virtual channel allocation and mapping manages the allocation of virtual channels and maps the virtual channel number of the input flits to its internal buffers. Observe that no crossbar switch is required. This simplifies the implementation of a router and makes it simpler and faster.

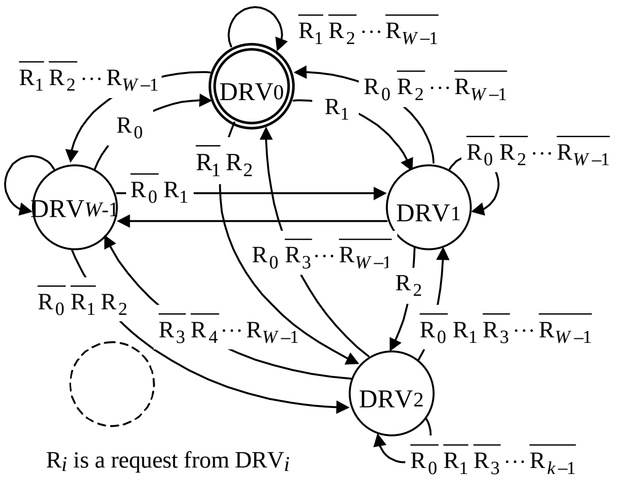
3.1 Physical Channel Arbitration

Recall that a multiway channel can have a maximum of *W* routers and nodes driving it. Only one router or node should be driving the channel at any given clock cycle. The physical channel arbitrator ensures exclusive access to the channel. Channel arbitration is a distributed hardware algorithm. All channel drivers apply the same algorithm, and thus the same decision is reached by all. The channel arbitrator must be fair to avoid starvation. Thus, it cannot assign fixed priorities to drivers.

The channel arbitrator determines which router (node) is driving a multiway channel at the current clock cycle and which router (node) will be driving the channel at the next cycle. At the beginning of each clock cycle, *W* request outputs are generated by the *W* drivers of a given channel. These *W* requests are then input to all the channel arbitrators through the arbitration lines. The channel arbitrators concurrently determine the next driver, based on the current driver and the input requests. The arbitration algorithm is a finite state machine as shown in Figure 6*a*. The current state of the FSM is the current driver and the transitions are based on the *W* requests: *R*0, *R*1, ..., and R_{W-1}, issued by the *W* drivers of a channel at a given clock cycle (the *ARB* lines in Figure 5 hold the *W* requests). The double-circled state in Figure 6*a* is the starting state on hardware reset.

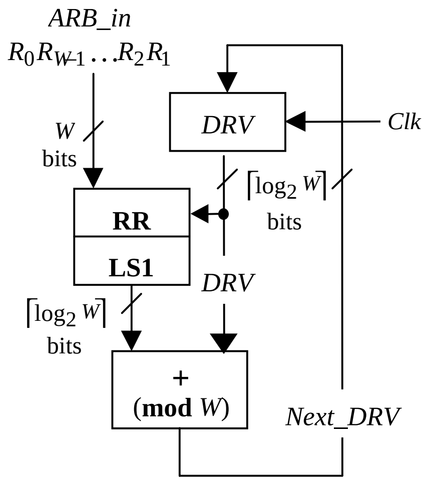
An implementation of the FSM of the channel arbitrator is the block diagram of Figure 6*b*. In this diagram, *DRV* is a register holding the number of the current driver and *Next_DRV* will be the driver number during the next clock cycle. *ARB_in* are the *W* input requests: $R_0, R_{W-1}, \cdots, R_2, R_1$ that are right rotated so that R_1 appears as the least significant bit, R_2 as the second least significant bit, and R_0 as the most significant bit. **RR** is a combinational circuit that does right rotation on the *ARB_in* bits by the current value of *DRV*. **LS1** is another combinational circuit that returns the address of the least significant bit whose value is 1. The address of the least significant bit is 1 (not 0), the address of the second least significant bit is 2, and the address of the most significant bit is 0 (not *W*-1). If no request is issued (i.e., all the request bits are zeros), **LS1** returns 0. The address of the least significant 1 is then added to *DRV* to obtain *Next_DRV*. The diagram of Figure 6*b* is effectively an implementation of the following equation:

$$Next\_DRV = \mathbf{LS1}(ARB\_in\ \mathbf{RR}\ DRV) + DRV\ \mathbf{MOD}\ W$$



DRV0
DRV1
DRV2
DRVW-1
$\overline{R_1}\,\overline{R_2}\cdots\overline{R_{W-1}}$
$\overline{R_1}\,\overline{R_2}\cdots R_{W-1}$
$R_0\,\overline{R_2}\cdots\overline{R_{W-1}}$
R_1
R_0
$\overline{R_1}\,R_2$
$\overline{R_0}\,\overline{R_2}\cdots\overline{R_{W-1}}$
$\overline{R_0}\,R_1$
$R_0\,\overline{R_3}\cdots\overline{R_{W-1}}$
R_2
$\overline{R_0}\,\overline{R_1}\,R_2$
$\overline{R_3}\,\overline{R_4}\cdots R_{W-1}$
$\overline{R_0}\,R_1\,\overline{R_3}\cdots\overline{R_{W-1}}$
$\overline{R_0}\,\overline{R_1}\,\overline{R_3}\cdots\overline{R_{k-1}}$
R_i is a request from DRV_i

Figure 6a: FSM of the Channel Arbitrator



ARB_in
$R_0R_{W-1}\ldots R_2R_1$
W bits
DRV
Clk
$\lceil \log_2 W \rceil$ bits
RR
LS1
DRV
$\lceil \log_2 W \rceil$ bits
+ (mod W)
Next_DRV

Figure 6b: Implementing the Channel Arbitrator

Consider an 8-way channel with *ARB_in* = *R*0 *R*7 *R*6 *R*5 *R*4 *R*3 *R*2 *R*1 = 10001011 and with *DRV* = 4 during the current cycle. Then, 10001011 **RR** 4 = 10111000 and **LS1**(10111000) = 4. Therefore, *Next_DRV* = 4+4 **MOD** 8 = 0. This is the expected answer, because *R*0 is the first non-zero request that appears after *R*4 (*R*5 = *R*6 = *R7* = 0).

A physical channel arbitrator generates an Output Enable signal (*OE* in Figure 5) when the current driver number *DRV* is equal to the driver identifier of that channel interface. The driver identifier is either *DIM*+ or *DIM*–, where the dimensionality, *DIM*, of the router is supplied directly through external pins. Since the driver identifier is unique in each driver of a given physical channel, the OE signal will be 1 in exactly one driver (router or node) at any given time. When the output is enabled, the channel interface puts the *DATA_out*, *TAG_out*, and *VC_out* signals on the physical channel. The channel interface uses three-state buffers to drive the physical channel.

Other approaches for physical channel arbitration do exist, but I believe that my approach is simple and fast. I should emphasize that physical channel arbitration and flit transfer are completely overlapped during a given clock cycle. No cycles are wasted just for channel arbitration. The number of *ARB* lines in a multiway channel should be *W*. This number can be reduced by half if only *W*/2 sources can request the channel in a given clock cycle.

3.2 Virtual Channels

Virtual channels allow the simultaneous multiplexing of different messages across a physical channel on a flit-by-flit basis. Virtual channels were originally introduced to solve the problem of deadlock in wormhole-routed networks. However, they are also used to reduce message latency and improve network throughput [Dally 92].

A virtual channel consists of a buffer that can hold one or more flits of a message and associated state information. Several virtual channels share the bandwidth of a multiway channel. Shown in Figure 5 are three virtual channel buffers per direction. However, a relatively large number (8 or 16) of virtual channel buffers can be supported quite easily.

When more than one virtual channel buffer exists per direction, a virtual channel arbitrator selects the next non-empty virtual channel buffer to drive the physical channel. To be fair and to avoid starvation, the virtual channel arbitrator uses a hardware algorithm, similar to one of the physical channel arbitrator, to select the next non-empty virtual channel buffer. However, in the presence of a header flit, the virtual channel arbitrator gives the header flit a higher priority if it has been successfully routed. Routing of header flits is discussed in Section 4.1.

3.3 Virtual Channel Allocation

Each virtual channel buffer has associated state information. The state information includes the allocation status (i.e., free or allocated), the current number of flits in the buffer (i.e., the used part of the buffer), the driver number, and the driver virtual channel number. When a header flit is received, a new virtual channel buffer is allocated. When the tail of a message is successfully transmitted, its virtual channel buffer is freed.

To efficiently utilize a multiway channel, the status of the virtual channel buffers in the *W* directions across the channel should be known to all drivers. The *STAT_out* line in Figure 5 represents the status of all virtual channel buffers in a given direction. *STAT_out* is 1 when at least one of the virtual channel buffers is free. It is 0 when all the virtual channel buffers are allocated. The *STAT_out* lines of the *W* interfaces across a mutliway channel are collectively called the *STAT* lines as shown in Figure 5. The *W STAT* lines are then fed as input (*STAT_in*) to all the *W* drivers of a multiway channel. The routing logic uses the *W STAT_in* lines to select a routing way for a header flit, as will be discussed next.

4 Routing

Routing in networks that use multiway channels is different than routing in networks that use dedicated point-to-point channels. In a conventional router with multiple input and output channels, routing of a header flit is done internally inside the router. The routing logic decides which output channel (physical and virtual) to select based on the availability of the output channels and on the destination address in the header flit. The routing logic also controls a crossbar switch that establishes simultaneous paths between input and output channels.

Routing across a multiway channel is a distributed hardware algorithm, done simultaneously by all the routers and nodes that are connected to a physical channel. Only one router (node) will accept an injected flit, while all other routers (nodes) will reject it. If the injected flit is invalid (with tag I) then all routers and nodes will reject it and nothing takes place. If the injected flit is valid (header, body, or tail) and has been accepted then the receiving router (node) acknowledges the transfer of this flit by driving the tag lines back to invalid. Thus, the tag lines are used to implement the handshaking protocol.

4.1 Routing a Header Flit

Routing a header flit is to determine a routing way for the flit. The routing way is determined based on the destination address, stored in the header, and on the status of the virtual channels in the *W* directions. If the routing logic (see Figure 5) can successfully determine a routing way for which at least one of the virtual channel buffers is free then this routing way is stored in the header flit as shown in Figure 4.

When a driver injects a header flit into a multiway channel, it also injects the driver's virtual channel number of that header flit. The routing way, encoded in the header flit, is compared with the directionality of all the channel interfaces across a multiway channel. Only one channel interface, whose directionality is equal to the routing way, will accept the header flit and all the other interfaces will reject it. The *TAG_in* bits will be *H* (Header) in one router (or node interface) and will be *I* (Invalid) in all other routers and node interfaces.

Once a header flit is accepted, the virtual channel allocation and mapping unit will allocate a new virtual channel buffer for this header flit (*TAG_in* = *H*). This is shown in Figure 7. In this Figure, *A* is the allocation bit (0=Free and 1=Allocated), *Drv* is the driver's number, and *VC* is the driver's virtual channel number. There should be at least one free virtual channel. Otherwise, the transfer of the header flit will not be acknowledged. The virtual channel allocation and mapping unit stores the driver's number (supplied by the physical channel arbitrator) and the driver's virtual channel number in the allocated entry. The allocated virtual channel number is then used to select the virtual channel buffer (*VCS* lines). The driver does not (and need not) known the virtual channel number of the allocated buffer.

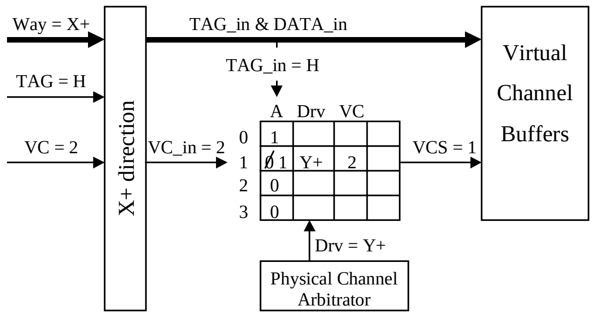


Figure 7: Allocating a virtual channel for a header flit

4.2 Routing a Body or a Tail Flit

When the tag of a flit on a multiway channel is *H* (Header), the *Way* encoded in the first few bits of the header flit will guide its routing. When the tag is *B* (body) or *T* (tail), all virtual channel allocation units across a multiway channel will lookup their tables concurrently. The allocation tables are searched by content for an allocated entry that holds the driver's number, supplied by the physical channel arbitrator, and the driver's virtual channel number given at the *VC* lines. This entry should be found in exactly one table (where it was previously allocated). Once the entry is found, the driver's virtual channel number (*VC_in*) is mapped to the allocated number. The allocated virtual channel number is then placed on the *VCS* lines to select the appropriate virtual channel buffer. Thus, the virtual channel allocation unit, used to allocate new virtual channel buffers for header flits, is also used as a mapping unit for body and tail flits. If the mapped virtual channel buffer is not full, the transfer of the body (or tail) flit is acknowledged.

4.3 Routing Algorithms

Networks with multiway channels can use the same routing algorithms developed for networks with dedicated (i.e., point to point) channels. The routing algorithm can be deterministic, such as the XY routing used in 2D meshes and the Dimension-Order routing used in high-dimensional meshes and hypercubes. It can be partially adaptive such as the Planar Adaptive routing proposed by Chien and Kim [ChienKim 92], or the West-First and other similar algorithms that use the Turn Model of Glass and Ni [GlassNi 92]. The routing algorithm can be fully adaptive, such as routing based on the concept of Virtual Networks [LinderHarden 91] or based on the concept of Dimension Reversal [DallyAoki 93].

The routing algorithm can also be described as being minimal or non-minimal. It should be deadlock and livelock-free. The routing logic, shown in Figure 5, implements the routing algorithm. It determines the routing way for header flits, based on the routing information (destination address and possibly other information) stored in the header flit and the allocation status of the virtual channels in all directions (the *W*-bit *STAT_in* input in Figure 5).

5. Performance Evaluation

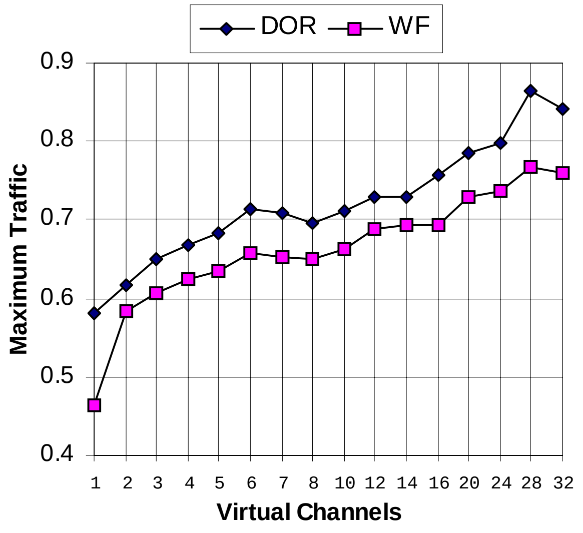
To measure the performance of interconnection networks with multiway channels, I have simulated a number of 2D mesh networks varying the length of messages, the number of virtual channels, the routing function, and the injection rate.

The simulator is a C++ program that simulates interconnection networks at the flit level. A flit transfer between two adjacent routers, over a multiway 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, different number of virtual channels and buffer sizes, different routing algorithms, and different traffic patterns. At the moment, the only topology supported is a 2D mesh because it is the representative of many parallel machines. This is the topology used in my simulation results. However, I am currently writing a newer version of the simulator that supports multi-dimensional meshes, tori, and multi-level trees. The impact of network topology on performance will be studied in another paper.

Simulations were run using dimension-order deterministic routing and also the turn model (west-first routing). *Latency* is measured by applying a constant rate source at each node and measuring the time from the injection of the header flit of a message at a source node until the ejection of the tail flit at a destination node. Source queuing time is not included in the latency measurement. The *network traffic* is the average number of flits transferred by *k*-way channels per clock cycle. The average is taken over all channels in the network and over a period of time. The network traffic is 100% (or 1) if all channels are busy transferring flits successfully at all times. In this situation, the network is said to run at full capacity.

The first experiment is to measure the maximum traffic that a 2D-mesh network can accept by varying the number of virtual channels at each router. The experiment uses dimension-order deterministic routing and west-first routing. The source nodes are driven at saturation to obtain the highest traffic possible for a given number of virtual channels. The traffic pattern is randomized. All messages are 64-byte long. The flit size is 16 bytes (128 bits). The results are shown in Figure 8*a*. Clearly, the network can accept more traffic as the number of virtual channel increases. However, with even 32 virtual channels per routing direction, the maximum network traffic was not able to reach 90% capacity! This may be attributed to the wormhole routing technique and is still under investigation. Finally, dimension-order routing is a clear winner over the west-first algorithm. This is because the network traffic is uniformly distributed and dimension-order routing works well with uniform traffic.

The second experiment is to measure the message latency by varying the injection rate and the number of virtual channels. The results are shown in Figure 8*b*. Dimension-order routing is used and all messages are 64-byte long. Observe that the graph is not a function. Both the latency and the traffic are measured values. When the traffic is below saturation, the message latency is not affected by the traffic. However, as the network saturates, latencies start increasing sharply. Driving the network hard beyond its saturation point deteriorates the traffic, but only slightly. Finally, adding virtual channels slightly increases the message latency for low traffic, but allows the network to accept more traffic.



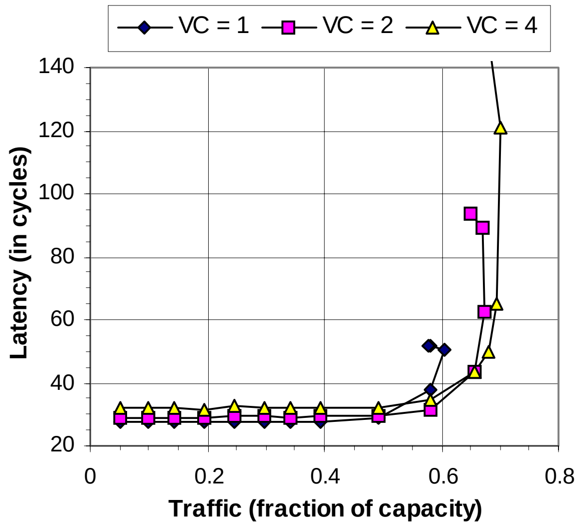


Figure 8a: Maximum traffic versus virtual channels
Figure 8b: Latency versus traffic (64-byte messages)

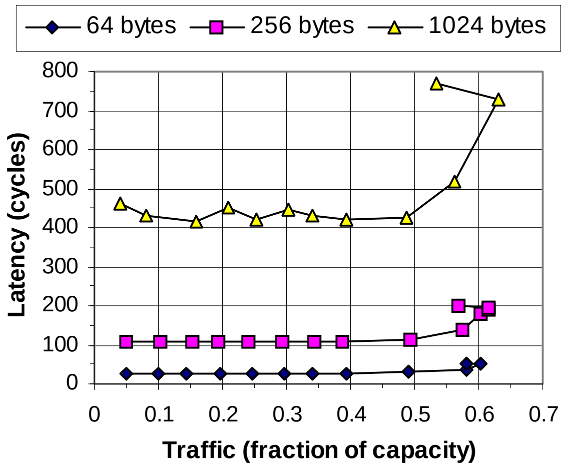


Figure 8c: Latency versus traffic under different message lengths

The last experiment is to measure the effect of message length on latency. Three message lengths are used in this experiment: 64 bytes, 256 bytes and 1024 bytes. The number of virtual channels was fixed to 1 and deterministic-order routing is used with uniformly distributed traffic. The results are shown in Figure 8*c*. The average latency of 64-byte messages is 28 cycles, the average latency of 256-byte messages is about 110 cycles, and the average latency of 1024-byte messages is about 440 cycles when the traffic is below saturation. The latency scaled linearly with the message length.

6. Conclusion and further research

This paper introduced a new class of interconnection networks based on the concept of *k*-way channels. The idea is to reduce the number of links per router to only two and to make channels very wide. A detailed design of a router was presented and the performance of 2D mesh networks was evaluated. The initial results are very encouraging and stimulate research in this direction. An improved version of the simulator is currently being developed to support different topologies and dimensions. We are also describing the router design in VHDL and implementing it using high-density FPGAs. Further research in this direction includes multicasting and broadcasting in networks with *k*-way channels, fault-tolerance, and the design of a versatile router that can be used to build many different topologies.

References

[Agarwal 90] A. Agarwal el al, "APRIL: A processor architecture for multiprocessing", *Proceedings of the 17th International Symposim on Computer Architecture*, pp 104-114, June 1990.

[Carbonaro 96] J. Carbonaro and F. Verhoorn, "Cavallino: The teraflops router and NIC", *Proceedings of Hot Interconnects Symposium IV*, August 1996.

[ChienKim 92] A. A. Chien and J. H. Kim, "Planar-adaptive routing: Low-cost adaptive networks for multiprocessors", *Proceedings of the 19th International Symposium on Computer Architecture*, pages 268-277, May 1992.

[Dally 94] W. J. Dally, et al., "Architecture and implementation of the Reliable Router", *Proceedings of Hot Interconnects Symposium II*, August 1994.

[Dally 92] W. J. Dally, "Virtual-channel flow control", *IEEE Transactions on Parallel and Distributed Systems*, vol.3, no.2, pages 194-205, March 1992.

[DallySeitz 87] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks", *IEEE Transactions on Computers*, vol. C-36, no.5, pages 547-553, May 1987.

[DallyAoki 93] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels", *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pages 466-475, April 1993.

[Duato 97] J. Duato et al, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, 1997.

[Kushkin 94] J. Kushkin et al, "The Stanford FLASH multiprocessor", *Proceedings of the 21st International Symposium on Computer Architecture*, pp 302-313, April 1994.

GlassNi 92] C. J. Glass and L.M. Ni, "The turn model for adaptive routing", *Proceedings of the 19th International Symposium on Computer Architecture*, pages 278-287, May 1992.

[Lenoski 92] D. Lenoski et al., "The Stanford DASH multiprocessor", *IEEE Computer*, vol 25, no 3, pp 63-79, March 1992.

[LinderHarden 91] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes", *IEEE Transactions on Computers*, vol. C-40, no. 1, pages 2-12, January 1991.

[Noakes 93] M. Noakes, D. Wallach, and W. Dally, "The J-Machine Multicomputer: An architecture evaluation", *Proceedings of the 20th International Symposimum on Computer Architecture*, pp 224-235, May 1993.

[Oed 93]W. Oed, "The Cray Research Massively Parallel Processing System: Cray T3D", *Cray Research*, 1993.

[Paragon 91] *Paragon XP/S Product Overview*, Intel Corporation, Supercomputer Systems Division, Beaverton, Oregon, 1991.

[ScottThorson 96] 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.