Virtual-Channel Flow Control

William J. Dally, *Member, IEEE*

Abstract— **Network throughput can be increased by dividing the buffer storage associated with each network channel into several virtual channels [11]. Each physical channel is associated with several small queues, virtual channels, rather than a single deep queue. The virtual channels associated with one physical channel are allocated independently but compete with each other for physical bandwidth. Virtual channels decouple buffer resources from transmission resources. This decoupling allows active messages to pass blocked messages using network bandwidth that would otherwise be left idle. This paper studies the performance of networks using virtual channels using both analysis and simulation. These studies show that virtual channels increase network throughput, by a factor of 4 for 10-stage networks, and reduce the dependence of throughput on the depth of the network.**

Index Terms— **Communication networks, concurrent computing, flow control, interconnection networks, multicomputers, multiprocessors, packet routing, parallel processing, virtual channels, wormhole routing.**

I. Introduction

Interconnection Networks

THE processing nodes of a concurrent computer exchange data and synchronize with one another by passing messages over an interconnection network [1], [2], [13], [4], [9], [24], [23]. The interconnection network is often the critical component of a large parallel computer because performance is very sensitive to network latency and throughput and because the network accounts for a large fraction of the cost and power dissipation of the machine.

An interconnection network is characterized by its topology, routing, and flow control [6]. The topology of a network is the arrangement of nodes and channels into a graph. Routing specifies how a packet 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. This paper deals only with flow control. It describes a method for allocating resources to packets using virtual channels [11]. This method can be applied to any topology and routing strategy.

Manuscript received January 15, 1991; revised May 24, 1991. This work was supported in part by the Defense Advanced Research Projects Agency under Contracts N00014-88K-0738 and N00014-87K-0825 and in part by a National Science Foundation Presidential Young Investigator Award Grant MIP-8657531, with matching funds from General Electric Corporation and IBM Corporation. A preliminary version of this paper appeared in the proceedings of the 17th International Symposium on Computer Architecture [8].

The author is with the Artificial Intelligence Laboratory and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

IEEE Log Number 9105524.

The Problem

The throughput of interconnection networks is limited to a fraction (typically 20%–50%) of the network's capacity [7] because of coupled resource allocation.

Interconnection networks are composed of two types of resources: buffers and channels. Typically, a single buffer is associated with each channel. Once a packet A is allocated a buffer b_i, no other packet B can use the associated channel c_i until A releases b_i. In networks that use flit[1]-level flow control [11], [1], [23], [9], packet A may be blocked due to contention elsewhere in the network while still holding b_i. In this case, channel c_i is idled even though there may be other packets in the network, e.g., packet B, that can make productive use of the channel.

This situation is illustrated in Fig. 1. In the figure, a fragment of a network is depicted with a rounded box denoting a node, a solid arrow a channel between two nodes, and a box denoting a flit buffer. Shaded arrows denote routes that are in progress. Packet A is blocked holding buffers 3E (east side of node 3) and 4S. Packet B is unable to make progress even though all physical channels it requires, (1E to 2W) through (4E to 5W), are idle because packet A holds buffer 3E which is coupled to channel (3E to 4W).

This problem of idling channels due to resource coupling is unique to interconnection networks that perform flow control at the flit-level. Most modern multicomputer networks that use circuit switching or wormhole routing [7] fall into this class. The problem does not occur in traditional packet-switched networks that perform flow control at the packet level since such networks never block a partially transmitted packet.

Virtual Channel Flow Control

A virtual channel consists of a buffer that can hold one or more flits of a packet and associated state information [11]. Several virtual channels may share the bandwidth of a single physical channel.[2]

Virtual channels decouple allocation of buffers from allocation of channels by providing multiple buffers for each channel in the network. If a blocked packet A holds a buffer b_{i0} associated with channel c_i, another buffer b_{i1} is available allowing other packets to pass A. Fig. 2 illustrates the addition of virtual channels to the network of Fig. 1. Packet A remains blocked holding buffers 3E.1 and 4S.1. In Fig. 2, however,

[1] A *flit* is a flow-control digit. See Section II-C for a more complete description.

[2] Virtual channels should not be confused with virtual circuits (named connections in a connection-oriented network [26], [3]) or with virtual cut-through (a packet-level flow-control technique [15]).

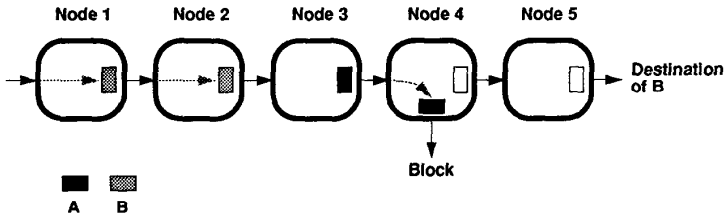


Fig. 1. Packet B is blocked behind packet A while all physical channels remain idle.

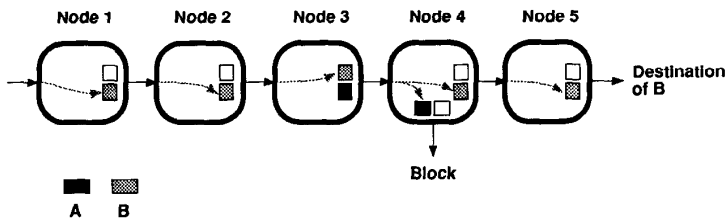


Fig. 2. Virtual channels provide additional buffers allowing packet B to pass blocked packet A.

Packet B is able to make progress because buffer 3E.2 is available allowing it access to channel (3E to 4W).

Adding virtual channels to an interconnection network is analogous to adding lanes to a street network. A network without virtual channels is composed of one-lane streets. In such a network, a single blocked packet blocks all following packets. Adding virtual channels to the network adds lanes to the streets allowing blocked packets to be passed.

In addition to increasing throughput, virtual channels provide an additional degree of freedom in allocating resources to packets in the network. This flexibility permits the use of scheduling strategies, such as routing the oldest packet first, that reduce the variance of network latency.

The most costly resource in an interconnection network is physical channel (wire) bandwidth. The second most costly resource is buffer memory. Adding virtual channel flow control to a network makes more effective use of both of these resources by decoupling their allocation. The only expense is a small amount of additional control logic.

Background

The use of virtual channels for flow control builds on previous work in using virtual channels for deadlock avoidance and in using output queueing or split input queues for partial resource decoupling. Virtual channels were introduced in [11] for purposes of deadlock avoidance. A cyclic network can be made deadlock-free by restricting routing so there are no cycles in the channel dependency graph and then adding virtual channels to reconnect the network. Virtual channels were first implemented for this purpose in the torus routing chip [10].

The network design frame [12] and the J-Machine network [9] use virtual channels to provide two logical networks on a single physical network. The iWARP processing element [4], [5] uses virtual channels (called logical channels in [4]) primarily to guarantee bandwidth to virtual circuits. iWARP virtual channels are sufficiently general that they can be used to decouple resource allocation as described in this paper.

A single stage of resource decoupling is provided by output queueing [14]. By performing the queueing in the output of a switch rather than the input, arriving packets are able to pass blocked messages arriving on the same input. Tamir [25] has shown how to achieve the same single-stage resource decoupling by partitioning the switch's input queue. This single stage resource decoupling is effective only if an entire packet fits in a single node. As shown in Fig. 3, When a packet too long to fit entirely in one input queue is blocked, it backs up into the output stage of the previous node preventing any following packet from passing it. With output queueing, there is still only a single output buffer associated with each physical channel. If a packet blocks while holding this output buffer, the channel is idled.

Our network analysis builds on the work of Patel [22] and of Kruskal and Snir [18] in analyzing unbuffered networks. We also build on the work of Kermani and Kleinrock [16] in analyzing buffered circuit switched, packet switched, and cut through networks without virtual channels. The analysis here extends this previous work by modeling the effect of virtual channels and by modeling networks with fixed sized buffers where packets are blocked (delay model) rather than dropped (loss model) when contention occurs.

Summary

The next section introduces the notation and assumptions that will be used throughout this paper. Section III describes virtual channel flow control in detail. An analysis of network performance is given in Section IV. The results of simulating networks using virtual channel flow control are described in Section V.

II. Preliminaries

A. Topology

An interconnection network consists of a set of *nodes*, N and a set of *channels*, $C \subseteq N \times N$. Each channel

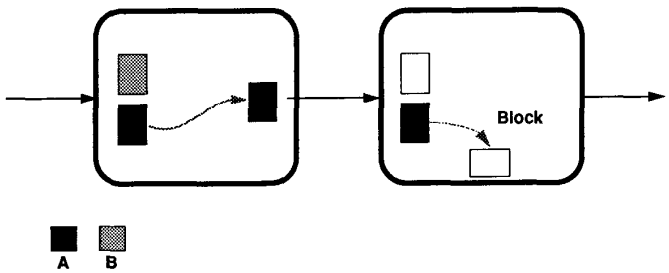


Fig. 3. Output queueing or partitioned input queues provide one stage of decoupling. However, long packets (such as packet A) continue to couple resources and cannot be passed.

is unidirectional and carries data from a source node to a destination node. A bidirectional network is one where $(n1, n2) \in C \Rightarrow (n2, n1) \in C$.

We have analyzed the performance of virtual channel flow control on multistage, k-ary n-fly, networks and have simulated the use of virtual channels on both multistage, networks and direct, k-ary n-cube, networks.

Multistage (k-ary n-fly) networks have k^n inputs connected to k^n outputs by n-stages of $k^{n-1}k \times k$-switches. For example, a 2-ary 4-fly is shown in Fig. 4.

A k-ary n-cube mesh network consists of k^n nodes arranged in an n-dimensional grid. Each node is connected to its Cartesian neighbors in the grid. For example, a 16-ary 2-cube is shown in Fig. 5.

The use of virtual channel flow control is in no way restricted to these two classes of networks. It is equally applicable to other topologies including trees, sorting networks, and irregular structures.

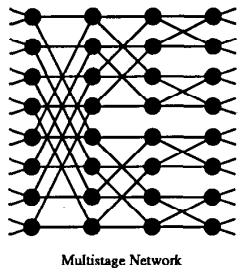


Fig. 4. A 2-ary 4-fly network.

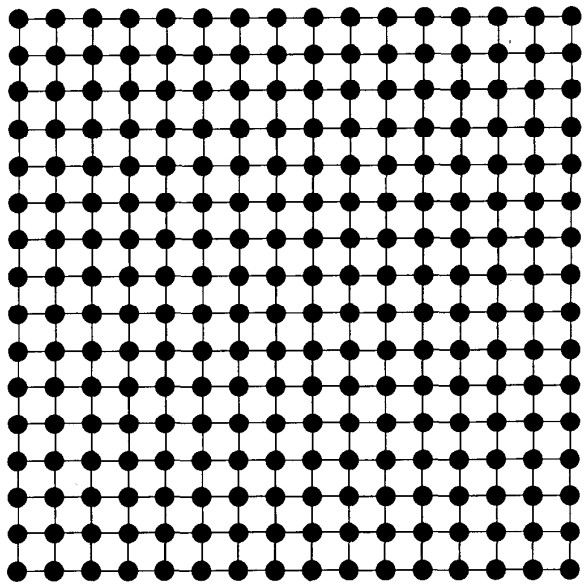


Fig. 5. A 16-ary 2-cube network.

B. Routing

A packet is assigned a route through the network according to a *routing relation*, $R \subseteq C \times N \times C$, given the channel occupied by the head of the packet and the destination node of the packet, the routing relation specifies a (possibly singleton) set of channels on which the packet can be routed.

C. Flow Control

Communication between nodes is performed by sending *messages*. A message may be broken into one or more *packets* for transmission. A packet is the smallest unit of information that contains routing and sequencing information. A packet contains one or more flow control digits or *flits*. A flit is the smallest unit on which flow control is performed. Information is transferred over physical channels in physical transfer units or *phits*. A phit is usually the same size or smaller than a flit.

The flow control protocol of a network determines 1) how resources (buffers and channel bandwidth) are allocated and 2) how packet collisions over resources are resolved. A resource collision occurs when a packet P is unable to proceed because some resource it needs (usually a buffer) is held by another packet. Collisions may be resolved by 1) blocking P in place, 2) buffering P in a node prior to where the collision occurs, 3) dropping P, or 4) misrouting P to a channel other than the one it requires. The technique described in this paper is applicable to all of these flow control strategies but is most appropriate for networks that use blocking or limited buffering to resolve collisions.

The flow control strategy allocates buffers and channel bandwidth to flits. Because flits have no routing or sequencing information, the allocation must be done in a manner that keeps the flits associated with a particular packet together. This may

be done by associating a set of buffers and some control state together into a virtual channel. A virtual channel is allocated to a packet and the buffers of the virtual channel are allocated in a FIFO manner to the flits of that packet. In the remainder of this paper, the terms *lane* and *virtual channel* are used interchangeably.

Most networks associate only a single virtual channel with each physical channel. This paper describes a method for improving the performance of networks by associating several virtual channels with each physical channel. This method makes no assumptions about how wires are allocated.

D. Wormhole Routing

The technique described here is particularly suitable for use in networks that use wormhole routing [7]. Wormhole routing refers to a flow-control protocol that advances each flit of a packet as soon as it arrives at a node (pipelining) and blocks packets in place when required resources are unavailable. Wormhole routing is attractive in that 1) it reduces the latency of message delivery compared to store and forward routing, and 2) it requires only a few flit buffers per node. Wormhole routing differs from virtual cut-through routing [15] in that with wormhole routing it is not necessary for a node to allocate an entire packet buffer before accepting each packet. This distinction reduces the amount of buffering required on each node making it possible to build fast, inexpensive routers.

III. Virtual Channel Flow Control

A. Structure

Each node of an interconnection network contains a set of buffers and a switch.[3] In this paper, we assume that the buffers are partitioned into sets associated with each input channel, an input-buffered node, as shown in Fig. 6. An output-buffered switch [14], [25] can be considered to be an input buffered switch with a nonblocking first stage by associating the buffers on the output of each stage with the inputs of the next stage.

A conventional network organizes the flit buffers associated with each channel into a first-in, first-out (FIFO) queue as shown in Fig. 7(a). This organization restricts allocation so that each flit buffer can contain only flits from a single packet. If this packet becomes blocked, the physical channel is idled because no other packet is able to acquire the buffer resources needed to access the channel.

A network using virtual channel flow control organizes the flit buffers associated with each channel into several lanes as shown in Fig. 7(b). The buffers in each lane can be allocated independently of the buffers in any other lane. This added allocation flexibility increases channel utilization and thus throughput. A blocked message, even one that extends through several nodes, holds only a single lane idle and can be passed using any of the remaining lanes.

[3] Each node also contains driver and receiver circuits to communicate across the physical wires and control logic.

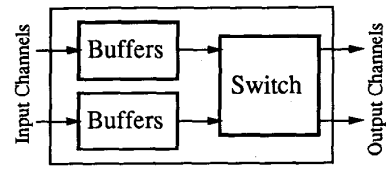


Fig. 6. Node organization. Each network node contains a set of buffers for each input channel and a switch.

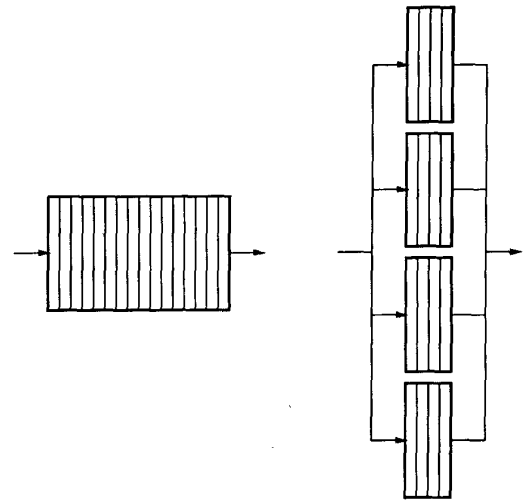


Fig. 7. (a) Conventional nodes organize their buffers into FIFO queues restricting routing. (b) A network using virtual-channel flow control organizes its buffers into several independent lanes.

B. Operation

In a network using virtual channel flow control, flow control is performed at two levels. Virtual channel assignment is made at the packet level while physical channel bandwidth is allocated at the flit level. When a packet arrives at a node, it is assigned (according to the routing algorithm) to an output virtual channel. This assignment remains fixed for the duration of the packet. The virtual channels associated with a physical channel arbitrate for physical channel bandwidth on a flit-by-flit basis.

Fig. 8 illustrates the hardware required to support virtual channel flow control on one physical channel. The transmitting node (node A) contains a status register for each virtual channel that contains the state of the lane buffer on the receiving node (node B). This state information includes: a bit to indicate if the lane is free, a count of the number of free flit buffers in the lane, and optionally the priority of the packet occupying the lane. node B contains a lane buffer and a status register for each virtual channel. The status maintained on node B includes input and output pointers for each lane buffer and the state of the channel: free, waiting (to be assigned an output), and active.

Lane assignment for physical channel P is performed by node A. When a packet arrives in an input buffer on node A

(not shown), it is assigned a particular output channel based on its destination, the status of the output channels, and the routing algorithm in use. The flow-control logic then assigns this packet to any free lane of the selected channel. If all lanes are in use, the packet is blocked in the waiting state until a lane is available. Maintaining lane state information on the transmitting end of the channel allows lane assignment to be performed on a single node. No additional internode communication is required to maintain this information as it is already required for flit-level flow control.

Once a lane is assigned to a packet, flit-level flow control is used to advance the packet across the switch and physical channel. To advance from an input buffer on the node A to an input buffer on node B, a flit must gain access to 1) a path through the switch to reach the output of node A, and 2) the physical channel to reach the input of node B. Typically either the switch is nonblocking, and thus always available (see Section III-D), or a few optional flit buffers are provided at the output of node A so that switch and channel resources do not have to be allocated simultaneously.

When the last flit of a message (the tail flit) leaves a node the lane assigned to that packet is deallocated and may be reassigned to another packet.

The status register storage required to implement virtual channel flow control for one physical channel, S_{vc} is shown below in terms of the number of lanes, l, the total number of flit buffers in the receiver, b, and the number of bits used to encode priority, pri. Setting $l = 1$ and pri $= 0$ gives the status register storage required by a conventional channel, S_{conv}. The first term of the expression corresponds to the storage on node A while the second term describes the storage on node B. For typical values of $b = 16$, $l = 4$, and pri$= 0$, S_{vc} is 36 bits compared with 17 bits for S_{conv}. This overhead is small compared to the storage required for the lane buffers, 512 bits if the flit size is 32 bits.

$$S_{\mathrm{vc}} = l\left(\lg\left(\frac{b}{l}\right) + 1 + \mathrm{pri}\right) + l\left(2\lg\left(\frac{b}{l}\right) + 2\right). \quad (1)$$

C. Allocation Policies

Flit-level flow control across the physical channel involves allocating channel bandwidth among lanes that 1) have a flit ready to transmit and 2) have space for this flit at the receiving end. Any arbitration algorithm can be used to allocate this resource including random, round-robin, or priority. For each physical channel, the arbitration algorithm is implemented as combinational logic that operates on the contents of the status registers and picks the highest priority lane that has space available at the receiving end. For random and round-robin arbitration schemes, priority information is generated by logic based on the lane's position and the previous state. For priority based schemes, priority information is stored in the status register for each lane.

Deadline scheduling [20] can be implemented by allocating channel bandwidth based on a packet's deadline or age—earliest deadline or oldest age first. Scheduling packets by age reduces the variance of message latency. Deadline scheduling provides several classes of delivery service and reduces the variance within each class.

D. Implementation Issues

Virtual channel flow control can be integrated into existing switch designs by replacing FIFO buffers with multilane buffers. When this replacement is made, however, the switch must be modified to deal with a larger number of inputs and outputs, and the flow control protocol between nodes must be modified to identify lanes.

Increasing the number of virtual channels multiplexed on each physical channel increases the number of inputs and outputs that must be switched at each node. If the switch handles each of these inputs and outputs separately as shown in Fig. 9(a), the switch complexity will increase significantly. Increasing the switch complexity is not required, however. The average data rate out of the set of lanes associated with a given physical channel is limited to the bandwidth of the channel. Thus it is sufficient to provide a single switch input for each physical input and output channel as shown in Fig. 9(b). With this organization, a small (one or two flit) output buffer is desirable to decouple switch allocation from physical channel allocation. Individual lanes are multiplexed onto the single path through the switch in the same manner that they are multiplexed over the single physical channel between the nodes.

An intermediate organization, shown in Fig. 9(c), is to provide a switch with separate inputs and multiplexed outputs. This configuration has the advantage of simple arbitration. An input virtual channel competes only for switch output ports. It need not simultaneously arbitrate for both input and output ports as is required for a fully multiplexed switch.

Any network that uses blocking or buffering flow control must, for each channel, send information in the reverse direction to indicate the availability of buffering on the receiving node. These acknowledgment signals can be transmitted on separate wires [12] or, in a bidirectional network, they can be transmitted out-of-band on a channel in the opposite direction [19].

In a network using multilane buffers, two effects increase the acknowledgment traffic. First, a few bits must be added to each acknowledgment signal to identify the lane being acknowledged. Second, because a lane buffer is typically smaller than a channel FIFO, the use of block acknowledgments to amortize the cost of the signal over several flits is restricted.[4]

Even with these effects, acknowledgment signal bandwidth is still a small fraction of forward bandwidth. In a network with 32-bit flits, 15 lanes per channel, and no block acknowledgment, 4 bits must be sent along the reverse channel for each flit transmitted along the forward channel, a 12.5% overhead. An additional 12.5% overhead is required to identify the lane associated with each flit sent in the forward direction. Such a scheme could be realized by a physical channel consisting of a 9-bit forward path (8-bit phits) and a 1-bit reverse path. Every four channel cycles a 32-bit flit is transmitted over the

[4] A block acknowledgment signals the availability of a block (several flits) of storage in a single action rather than signaling each flit separately

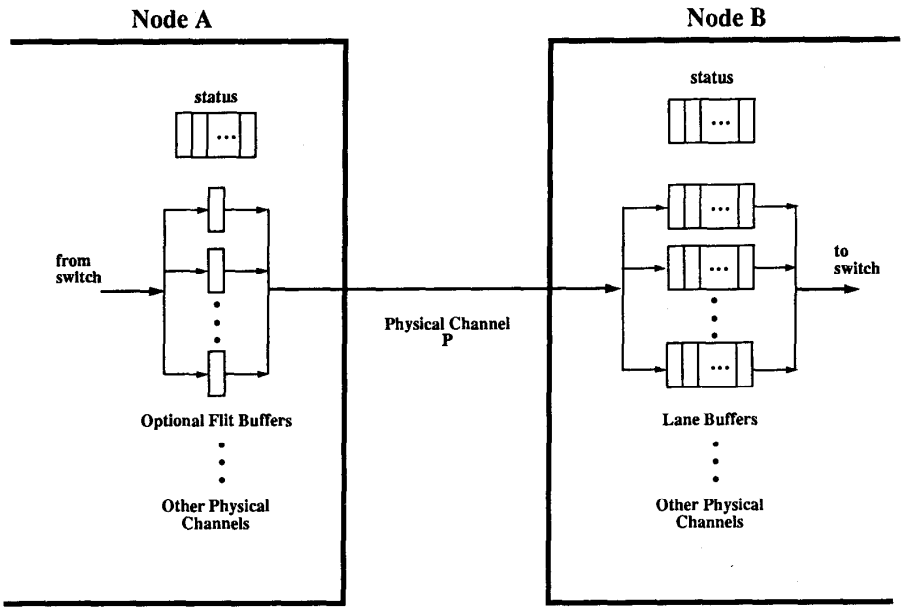


Fig. 8. Logic associated with one physical channel P to support virtual channel flow control. The transmitting node (node A) includes status registers for each virtual channel and optional flit buffers. The receiving node (node B) contains buffers and status registers.

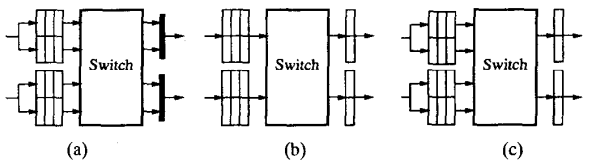


Fig. 9. (a) Adding virtual channels increases switch complexity if a complete switch is used. (b) Using a multiplexed switch leaves switch complexity unchanged. (c) Multiplexing only switch outputs give intermediate complexity and results in simpler arbitration than (b).

forward path along with its 4-bit lane identifier and a 4-bit acknowledgment code is transmitted over the reverse path.

IV. Analysis

This section develops an analytical performance model for k-ary n-fly networks that use virtual channel flow control.

For convenience, the following is a summary of the notation used in this section.

$b_{i,j}$ The probability that all occupied virtual channels are blocked in stage i when j virtual channels are occupied.

j_{avg} The expected number of virtual channels sharing a physical channel–averaged over packets.

l Number of virtual channels per physical channel.

L Message length in bits.

λ The offered traffic rate.

$\lambda_{\max}$ Network throughput, the maximum rate for which a steady state solution exists.

n Number of stages in the network.

$p_{i,j}$ The probability that j virtual channels are occupied in stage i.

$q_{i,j}$ An intermediate variable used in the calculation of $p_{i,j}$.

t_0 The service time at the destination.

$t_{i,j}$ The effective service time out of stage i when j virtual channels are occupied.

T Network latency.

T_d Time required for a flit to propagate through one node.

w_i The average total amount of time spent acquiring virtual channels between stage i and the destination inclusive.

W Channel width in bits.

x_i The average amount of time spent waiting to acquire a virtual channel at stage i.

We make the following assumptions

1) Packet destinations are uniformly randomly distributed.
2) A packet that arrives at its destination is consumed without waiting.
3) All packets are of length L.
4) At each source packets are created by a Poisson process with rate λ.
5) Each virtual channel is associated with a single flit buffer.
6) Packet blocking probabilities are independent.

The analysis considers a single path through the network. Our analysis starts at the destination and works back to the source. Network stages are numbered starting from the destination. The final stage is stage 0. The stage connected to the source is stage $n-1$.

The destination, stage 0, is always able to accept a packet, so the service time seen by a packet in the final stage is $t_0 = L/W$ which we will normalize to be unit time, $t_0 = 1.0$.

The service time at internal stages is increased because a channel may be idled when all of the virtual channels at a subsequent stage of the network are occupied. The average

amount of time that a packet waits to acquire a virtual channel at stage i of the network, x_i, is given by the product of the probability of all virtual channels in stage i being occupied, $p_{i,l}$, and the average waiting time, $t_{j,l}/2$.

$$x_i = \frac{p_{i,l} t_{j,l}}{2}. \tag{2}$$

The total time spent waiting to acquire virtual channels between stage i and the network output is the summation of the acquisition delays at each stage.

$$w_i = \sum_{j=0}^{i} x_j. \tag{3}$$

If j virtual channels at stage i are occupied, the total time required to service all j channels with no idling is jt_0. Each channel is individually blocked for a total time of w_{i-1}, so the probability of an individual channel being blocked is approximately[5] w_{i-1}/jt_0. Assuming that the blocking probabilities are independent, the probability of the channel being idled because all virtual channels are blocked, $b_{i,j}$, is the product of the individual probabilities.

$$b_{i,j} = \left(\frac{w_{i-1}}{jt_0}\right)^j. \tag{4}$$

Assuming independence of channel blocking probabilities is an approximation that slightly underestimates $b_{i,j}$. There is some dependence between blocking probabilities as it is possible for two packets to be blocked waiting on the same channel. This approximation, however, is justified because 1) it gives a result that agrees closely with experiment and 2) a calculation of $b_{i,j}$ that accounted for dependence did not give appreciably different results.

The effective service time seen by a packet at stage i with j virtual channels occupied, $t_{i,j}$, is thus

$$t_{i,j} = t_0(1 + b_{i,j}). \tag{5}$$

Equation (2) uses the probability of all l virtual channels being occupied. We calculate the occupancy probabilities for the virtual channels in a stage, i, using a Markov model as illustrated in Fig. 10. State S_j corresponds to j virtual channels being occupied. This state transitions to $j+1$ at rate λ, and to state $j-1$ at rate $1/t_{i,j}$. The rate out of the last stage is reduced to account for the arrival of packets while the stage is in this state. Solving this model for the steady state probabilities gives

$$q_{i,j} = \begin{cases} 1 & \text{if } j = 0 \\ q_{i,j-1}\lambda t_{i,j} & \text{if } 0 < j < l \\ q_{i,l-1}\frac{\lambda}{\frac{1}{t_{i,l}} - \lambda} & \text{if } j = l \end{cases} \tag{6}$$

$$p_{i,j} = \begin{cases} \frac{1}{\sum_{m=0}^{l} q_{i,m}} & \text{if } j = 0 \\ p_{i,j-1}\lambda t_{i,j} & \text{if } 0 < j < l \\ p_{i,l-1}\frac{\lambda}{\frac{1}{t_{i,l}} - \lambda} & \text{if } j = l. \end{cases} \tag{7}$$

[5]This is a slight overestimate as we are not considering the increase in effective service time due to blockage. The actual expression should be $w_{i-1}/(jt_0(1 + b_{i,j}))$ which results in a set of equations that cannot be solved in closed form for $j > 3$.

Network latency has four components as shown in (8).

$$T = \frac{L}{W}\left(j_{\text{avg}} + w_n + \frac{\lambda(t_0 + x_{n-1})^2}{2(1 - \lambda(t_0 + x_{n-1}))}\right) + T_d n. \tag{8}$$

The first term accounts for multiplexing delay due to virtual channels. If in the most congested stage j_{avg} virtual channels are occupied on average, a packet takes $j_{\text{avg}}L/W$ time to entirely traverse one channel. As the first stage, stage $n-1$, is always the most congested, j_{avg} can be calculated by

$$j_{\text{avg}} = \frac{\sum_{j=1}^{l} j^2 p_{n-1,j}}{\sum_{j=1}^{l} j p_{n-1,j}}. \tag{9}$$

The second term, $w_n L/W$, is the time spent waiting to acquire virtual channels.

The third term is the queueing delay in the source which is that of an M/D/1 queue with utilization $\rho = \lambda(t_0 + x_{n-1})$ and average service time $\overline{x} = L/W(t_0 + x_{n-1})$ [17]. For heavily loaded networks this term is the most significant. Modeling the source queue as M/D/1 is a slight approximation since even though message length is fixed the service time seen by the source has nonzero variance.

The final term, $T_d n$, is the time one flit of a packet spends traversing the n stages of the network in the absence of contention.

Fig. 11 shows the latency predicted by (8) as a function of offered traffic, λ, for a 2-ary 8-fly network. The figure shows that the addition of virtual channels greatly increases the traffic that can be offered to the network before saturation occurs. The figure also indicates that adding lanes has little effect on latency below saturation throughput. The curves lie on top of each other until traffic approaches saturation throughput.

Using virtual channel flow control increases saturation throughput because it decreases both waiting time and physical channel idle time. Waiting time, w_{n-1} (3), is reduced because a packet waits only when all l virtual channels are occupied. In a conventional network, $l = 1$, the blocking probability $p_{i,l}$ is much larger and w_{n-1} is increased proportionally. Waiting time is particularly important since it determines the service time seen by the source queue and hence the source queueing delay.

Physical channel idle time is reduced because a physical channel idles only when all of its virtual channels are blocked. Because virtual channels make this blocking probability, $b_{i,j}$ (4), small, the effective service time increases more slowly with traffic than in a conventional network.

The throughput $\lambda_{\max}$ of a multistage network using virtual channels can be determined by solving the equation

$$\lambda_{\max} = \frac{1}{t_0 + x_{n-1}}. \tag{10}$$

Fig. 12 shows the throughput of several multistage networks as a function of the number of virtual channels, l. The figure shows that adding a small number of virtual channels causes a large increase in throughput with diminishing returns as more channels are added. The data suggests that four to eight lanes per physical channel is adequate for most networks. For all of

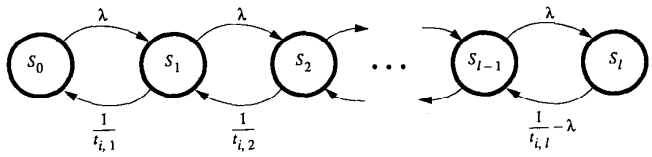


Fig. 10. Virtual channel occupancy probabilities are calculated using a Markov model. State S_j corresponds to j virtual channels being occupied.

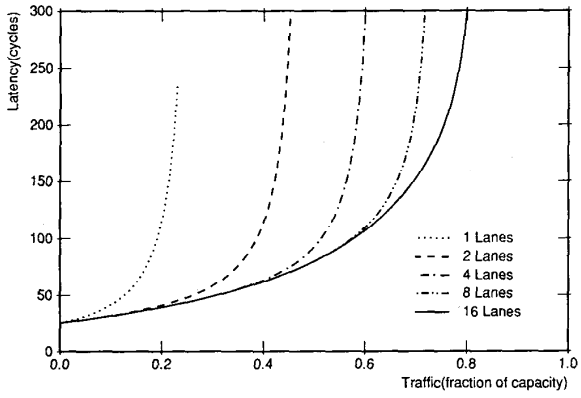


Fig. 11. Latency as a function of offered traffic as predicted by analysis for a 2-ary 8-fly network with 1, 2, 4, 8, and 16 virtual channels per physical channel.

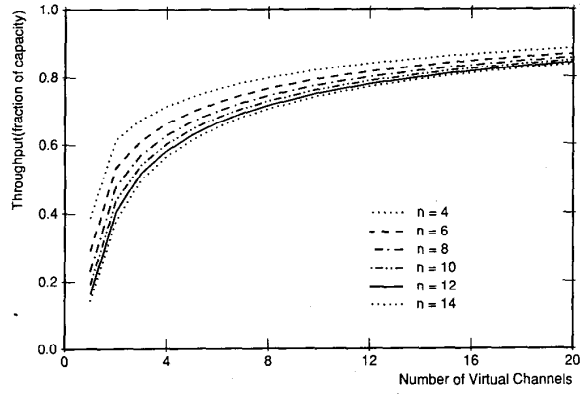


Fig. 12. Throughput of 2-ary n-cube networks with virtual channels as a function of the number of virtual channels.

the networks shown, eight lanes per physical channel results in at least 80% of the throughput as a network with 20 lanes.

Adding virtual channels to a network also reduces the dependence of throughput on the number of stages in the network. For conventional networks, $l = 1$ on the left side of the figure, there is a large difference in throughput with network size. Conventional networks with four and fourteen stages have throughputs of 0.39 and 0.14, respectively. With 18 virtual channels, the difference is narrowed to 6%, 0.83 versus 0.88. The effect of the number of stages on throughput is reduced by virtual channels because the reduction in waiting time and channel idle time reduce the rate at which service time increases with stage number.

V. Experimental Results

To measure the effect of virtual channel flow control on network performance (throughput and latency), we have simulated a number of k-ary n-cube and k-ary n-fly networks. These simulations serve to check our analytical model, presented in Section IV, and to measure the performance of networks not covered by our model.

Our analytical model is limited to networks where lanes have unit depth independent of the number of virtual channels. We have simulated networks where the total buffer storage per physical channel is held constant constant while varying the number of lanes per channel. If lanes are added, the depth of each lane is proportionally reduced. These simulations compare the effect of increasing the number of virtual channels with increasing the depth of each virtual channel.

We have also simulated k-ary n-cube networks with virtual channels, a topology not covered by our analytic model, and networks employing deadline and priority scheduling.

The simulator is a 3000 line C program that simulates interconnection networks at the flit-level. A flit transfer between two nodes is assumed to take place in one time unit. The network is simulated synchronously moving all flits that have been granted channels in one time step and then advancing time to the next step. The simulator is programmable as to topology, routing algorithm, and traffic pattern.

The simulations were run with packet length fixed at 20 flits and uniformly distributed random packet destinations. Except where otherwise noted, channel bandwidth was allocated randomly to lanes.

Each simulation was run for a total of 30 000 flit times. Statistics gathering was inhibited for the first 10 000 flit times to avoid distortions due to the startup transient. For a typical test, ($\lambda = 0.4$, $n = 8$), measurements were taken for 100 000 packets. The standard deviation of both latency and throughput measurements for an individual packet is $\approx$ 30% of the mean value. Assuming that these values are independent and normally distributed, the standard deviation of the ensemble average measurements reported below is $\approx 0.1\%$ of the mean value.

A. Throughput

Throughput is measured by applying to each network input a saturation source that injects a new packet into the network whenever a lane is available on its input channel. Throughput is given as a fraction of network capacity. A uniformly loaded network is operating at capacity if the most heavily loaded channel is used 100% of the time.

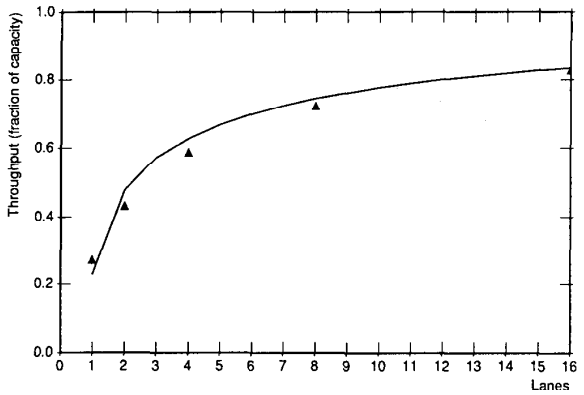


Fig. 13. Comparison of predicted (solid line) and measured (points) throughput for radix-2 eight stage networks.

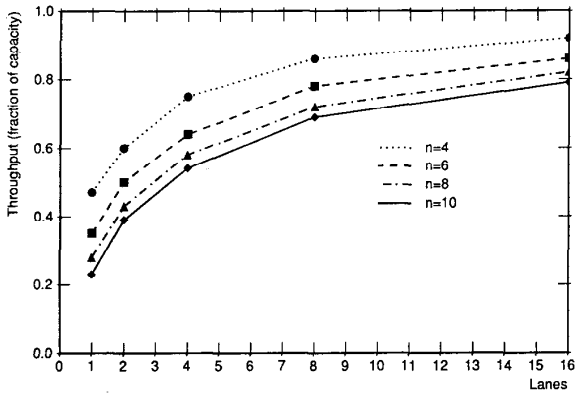


Fig. 14. Throughput versus number of lanes for radix-2 multistage networks holding the amount of storage per physical channel constant at 16 flits.

Fig. 13 compares the throughput predicted by the analytic model of Section IV with measurements from the simulator. The figure shows the throughput of 2-ary 8-fly networks with a single flit buffer per virtual channel as the number of virtual channels (lanes) per physical channel are varied. There is close agreement between the predicted and measured data.

To compare the effect of adding lanes to the network with adding depth to each lane, networks with equal storage were simulated. Each network simulated has 16 flits of storage per physical channel. The number of lanes per channel was varied from 1 (conventional network) to 16 in powers of two. The results of these simulations are shown in Fig. 14. The figure shows the saturation throughput versus the number of lanes per channel for radix-2 multistage networks (2-ary n-flys). Data are shown for networks with dimensions of 4, 6, 8, and 10. The data show that given a fixed amount of storage, adding lanes gives a far greater throughput improvement than does increasing the total amount of buffering with a single lane (see [21]).

B. Latency

Latency is measured by applying a constant rate source with exponentially distributed interarrival times to each input and measuring the time from packet creation until the last flit of the packet is accepted at the destination. Source queueing time is included in the latency measurement.

Fig. 15 compares the latency predicted by the model of Section IV with measurements from the simulator. The figure shows the latency of a 2-ary 8-fly network with four virtual channels per physical channel. Each virtual channel has a single flit buffer. As with the throughput comparison shown in Fig. 13, the latency measurements are in close agreement with the value predicted by analysis.

Fig. 16 shows latency results for 2-ary 8-fly networks with equal storage. Each network simulated has 16 flits of storage per physical channel. The number of lanes per node was varied from 1 to 16 in powers of two. As with the throughput results, Fig. 16 shows that adding lanes to a network is a more effective use of storage than adding depth to a lane. However, with

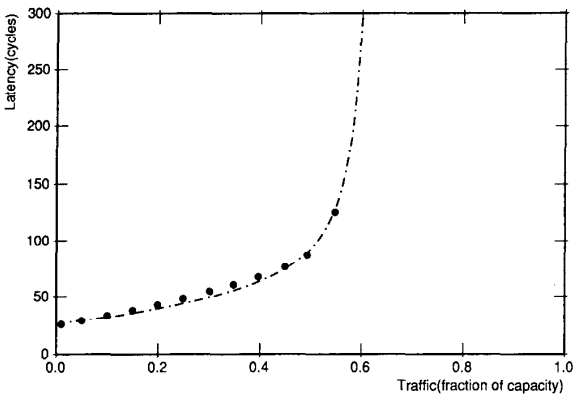


Fig. 15. Comparison of predicted (solid line) and measured (points) latency for a 2-ary 8-fly network with four lanes.

storage held constant, there is a small latency penalty (as much as 7%) at mid to high traffic rates associated with the shorter buffers that result when more virtual channels are used. For example, the curve for eight lanes drops below the curve for 16 lanes until it nears its saturation throughput.

Adding lanes is more effective in increasing throughput than adding depth for two reasons. First, adding lanes reduces the channel blocking probability exponentially, (4), while adding depth has a very small effect on blocking probability. Second, adding a single flit virtual channel buffer allows one packet to pass another. This has an effect on waiting time comparable to increasing the depth of a buffer by the packet length.

C. Scheduling Algorithm

Fig. 17 shows the effect of the channel scheduling algorithm on latency. The figure shows two latency histograms, one for a random assignment of channel bandwidth to packets, and the other for oldest-packet-first channel bandwidth allocation (deadline scheduling). Both curves are for 2-ary 6-fly networks with random traffic operating at 50% capacity. The histograms have been truncated at 140 cycles latency.

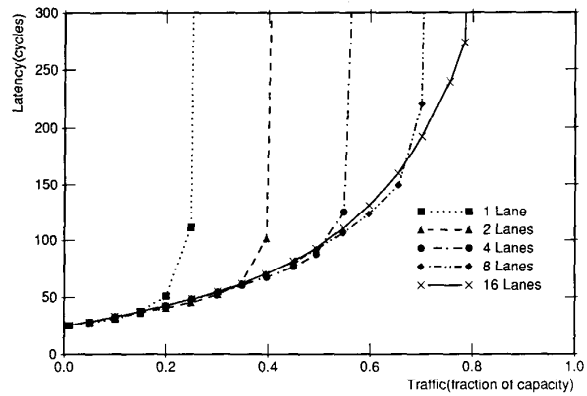


Fig. 16. Simulated latency versus offered traffic for 2-ary 8-fly networks with 16 flits of storage per physical channel.

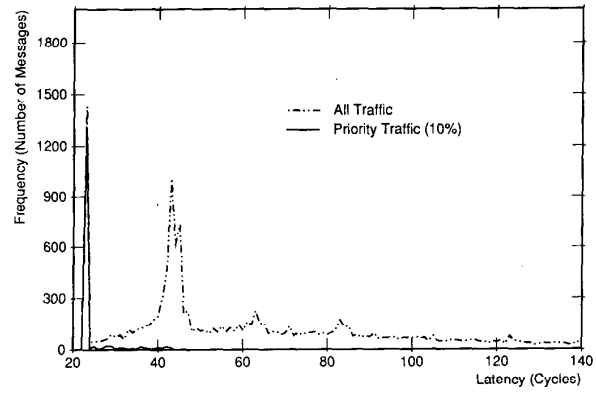


Fig. 18. Latency histogram for a 2-ary 6-fly network with random traffic at 50% capacity where 10% of the traffic is marked high-priority and scheduled by age. 80% of the high-priority traffic is delivered with the minimum latency.

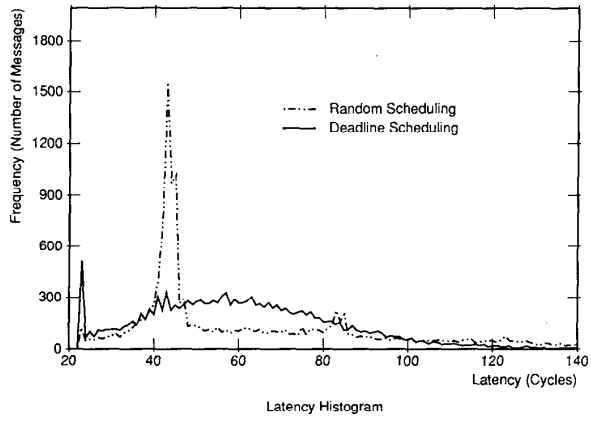


Fig. 17. Latency histogram for a 2-ary 6-fly network with random traffic at 50% capacity using deadline (oldest first) and random scheduling of physical channels. The histogram for deadline scheduling has a very sharp peak at 24 and a broad peak at 57. The curve for random scheduling shows a peak at 44.

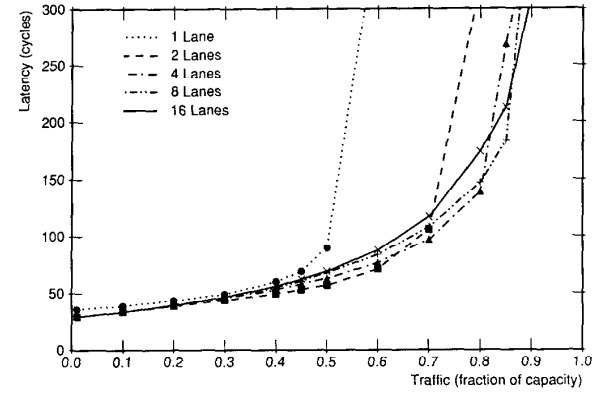


Fig. 19. Latency versus offered traffic for 16-ary 2-cube mesh networks under random traffic. As with the butterfly networks, adding lanes to a network has little effect on latency below the saturation throughput for a single lane. Adding a lane dramatically increases throughput at first with diminishing returns for 8 and 16 lanes.

The use of deadline scheduling reduced the average packet latency from 75.1 cycles to 62.2 cycles and reduced the standard deviation of packet latency by a factor of two. The deadline curve shows a sharp peak at 24 cycles latency, the minimum latency required to traverse the network, and a broad peak at 57 cycles. The random curve shows peaks at 44 and 84 cycles. The data suggest that deadline scheduling can be useful in reducing average message latency and in making message latency more predictable.

Fig. 18 shows how virtual channel scheduling can be used to provide different classes of service. The figures shows the result of an experiment where a network was loaded with 90% standard traffic and 10% priority traffic. The standard traffic was scheduled randomly, the priority traffic took precedence over standard traffic and was scheduled by age—oldest packet first. The figure shows that 80% of the priority traffic is delivered with the minimum latency (24 cycles). This type of scheduling would be useful, for example, in a switch that handles both voice and data where the voice traffic has a tight deadline and should be dropped if it cannot make its deadline.

D. k-ary n-cube Simulations

Fig. 19 shows average packet latency as a function of offered traffic for radix-16, dimension-2 mesh networks (16-ary 2-cubes). Each network simulated has 32 flits of storage per physical channel. The number of lanes per channel was varied from 1 to 16 in powers of two. The simulated network uses deterministic dimension-order routing.

The figure shows that adding lanes has little effect on latency below the saturation throughput for a single lane while it greatly increases the saturation throughput of the network. With a single lane, the network saturates at 50% capacity while with 16-lanes throughputs of 90% capacity are achievable. Most of the throughput gain is realized with four lanes. Adding additional lanes yields diminishing returns and actually increases latency by as much as 20%. As with the k-ary n-fly, latency in the mid to high traffic range is increased slightly by adding lanes. We suspect that this increase occurs because adding lanes while holding storage constant increases buffer utilization.

Direct, k-ary n-cube, networks using dimension-order routing require fewer virtual channels than multistage networks because most contention in these networks occurs when packets enter a new dimension. For purposes of analyzing contention, each dimension of a direct network is analogous to a stage of a multistage network. With few dimensions, there are fewer opportunities for blocking and fewer lanes are required to pass blocked packets.

VI. Conclusion

The performance of interconnection networks can be improved by organizing the buffers associated with each network channel into several lanes or virtual channels rather than a single FIFO queue. Associating several lanes with each physical channel decouples the allocation of virtual channels to packets from the allocation of physical channel bandwidth to flits. This decoupling allows active messages to pass blocked messages dramatically improving network throughput.

The use of virtual channel flow control also allows flexibility in allocating physical channel bandwidth. By decoupling resource allocation, a channel's bandwidth need not be allocated to the "next packet in line." Instead, this bandwidth may be allocated on the basis of packet type, age, or deadline. The use of deadline scheduling may be particularly important in networks where one class of packets must be delivered in a bounded amount of time.

This paper has developed a model for the performance of k-ary n-fly networks with virtual channels. This model can be used to calculate the latency and throughput of these networks. The model shows that adding virtual channels to a network greatly improves network throughput with little effect on latency at low traffic rates. For a 2-ary 10-fly network (1024 input butterfly), the throughput with 16 lanes per channel is 4 times the throughput with a single lane. The use of virtual channels also reduces the dependence of throughput on the number of stages in a network. Without virtual channels, throughput asymptotically varies as the inverse of the number of stages [18]. With virtual channels, throughput remains relatively constant as stages are added.

Several indirect (k-ary n-fly) and direct (k-ary n-cube) networks have been simulated to validate our model and to study networks not covered by the model. Simulation results agree closely with the values predicted by our model. Simulations also show that with the total amount of buffer storage per node held constant, adding lanes to a network is a significantly more effective use of storage than adding depth to a channel FIFO. The use of deadline scheduling reduces average latency by a small amount and makes latency much more predictable.

The critical resources in an interconnection network are wire bandwidth and buffer memory. Virtual channel flow control is a method for allocating these critical resources in a more efficient manner. With network switches constructed using VLSI circuits, the cost of adding the small amount of control state and logic required to implement multiple lanes per channel is well worth the cost.

Acknowledgment

I thank S. Ward, A. Agarwal, T. Knight, and C. Leiserson for many discussions about interconnection networks and their analysis. I thank all the members of the MIT Concurrent VLSI Architecture group and especially S. Wills, D. Chaiken, W. Horwat, and H. Aoki for their help with and contributions to this paper. L. Sardegna deserves thanks for preparing several of the illustrations in this paper. I thank the editor, B. Buckles, for arranging a speedy review of this manuscript and the referees (especially referee 4) for many helpful comments and suggestions.

References

[1] R. Arlauskas, "iPSC/2 system: A second generation hypercube," in *Proc. Third Conf. Hypercube Concurrent Comput. and Appl.*, ACM, 1988, pp. 33–36.

[2] W. C. Athas and C. L. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Comput. Mag.*, vol. 21, pp. 9–24, Aug. 1988.

[3] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[4] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iWARP: An integrated solution to high-speed parallel computing," in *Proc. Supercomput. Conf.*, IEEE, Nov. 1988, pp. 330–338.

[5] S. Borkar *et al.*, "Supporting systolic and memory communication in iWarp," in *Proc. 17th Int. Symp. Comput. Architecture*, May 1990, pp. 70–81.

[6] W. J. Dally, "Network and processor architecture for message-driven computers," in *VLSI and Parallel Computation*, Suaya and Birtwhistle, Eds. Los Altos, CA: Morgan Kaufmann, 1990.

[7] ——, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Comput.*, vol. 39, June 1990. Also appears as a chapter in *Artificial Intelligence at MIT, Expanding Frontiers*, edited by P. H. Winston, with Sarah A. Shellard, vol. 1. Cambridge, MA: MIT Press, 1990, pp. 548–581.

[8] ——, "Virtual-channel flow control," in *Proc. 17th Annu. Int. Symp. Comput. Architecture*, Los Alamitos, CA, IEEE Computer Society Press, May 1990, pp. 60–68.

[9] W. J. Dally *et al.*, "The J-Machine: A fine-grain concurrent computer," in *Proc. IFIP Congress*, G. X. Ritter, Ed. New York: North-Holland, Aug. 1989, pp. 1147–1153.

[10] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Comput.*, vol. 1, pp. 187–196, 1986.

[11] ——, "Deadlock free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, pp. 547–553, May 1987, VLSI memo 87–417.

[12] W. J. Dally and P. Song, "Design of a self-timed VLSI multicomputer communication controller," in *Proc. Int. Conf. Comput. Design*, IEEE Computer Society Press, Oct. 1987, pp. 230–234.

[13] BBN Advanced Computers Inc., "Butterfly parallel processor overview," BBN Rep. 6148, Mar. 1986.

[14] M. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Commun.*, vol. COM-35, Dec. 1987.

[15] P. Kermani and L. Kleinrock, "Virtual cut through: A new computer communication switching technique," *Comput. Networks*, vol. 3, pp. 267–286, 1979.

[16] ——, "A tradeoff study of switching systems in computer communication networks," *IEEE Trans. Comput.*, vol. C-29, pp. 1052–1060, Dec. 1980.

[17] L. Kleinrock, *Queueing Systems, Vol. 1: Theory*. New York: Wiley, 1975.

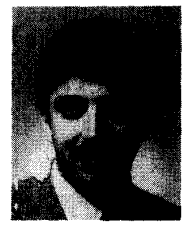
[18] C. P. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Trans. Comput.*, vol. C-32, pp. 1091–1098, Dec. 1983.

[19] InMOS Ltd., *IMS T424 Reference Manual*, Order Number 72 TRN 006 00, Nov. 1984.

[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[21] J. N. Mailhot, "Routing and flow control strategies in multiprocessor networks," S.B. thesis, May 1988.

[22] J.H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.*, vol. C-30, pp. 771–780, 1981.
[23] C.L. Seitz *et al.*, "The architecture and programming of the Ametek Series 2010 multicomputer," in *Proc. Third Conf. Hypercube Concurrent Comput. and Appl.*, ACM, 1988, pp. 33–36.
[24] C.L. Seitz, "The Cosmic Cube," *Commun. ACM*, vol. 28, pp. 22–33, Jan. 1985.
[25] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communication switches," in *Proc. 15th Annu. ACM/IEEE Int. Symp. Comput. Architecture*, June 1988, pp. 343–354.
[26] A.S. Tanenbaum, *Computer Networks*, second ed. Englewood Cliffs, NJ: Prentice-Hall 1988.



William J. Dally (S'78–M'86) received the B.S. degree in electrical engineering from Virginia Polytechnic Institute, the M.S. degree in electrical engineering from Stanford University, and the Ph.D. degree in computer science from Caltech.

He has worked at Bell Telephone Laboratories where he contributed to the design of the BELLMAC-32 microprocessor. Later as a consultant to Bell Laboratories he helped design the MARS hardware accelerator. He was a Research Assistant and then a Research Fellow at Caltech where he designed he MOSSIM Simulation Engine and the Torus Routing Chip. He is currently an Associate Professor of Computer Science at the Massachusetts Institute of Technology, Cambridge, where he directs a research group that is building the J-Machine, a fine-grain concurrent computer. His research interests include concurrent computing, computer architecture, computer aided design, and VLSI design.