Design and Evaluation of a DAMQ Multiprocessor Network With Self-Compacting Buffers

J. Park
IBM Corporation.
Appl. Business Sys.
Endicott, NY 13760

B. W. O'Krafka
IBM Corporation
Adv. Wkstation Div.
Austin, TX 78758

S. Vassiliadis
Dept. of Elctr. Engr.
Delft Univ. of Tech.
The Netherlands

J. Delgado-Frias
Dept. of Elctr. Engr.
SUNY at Binghamton
Binghamton, NY

Abstract

This paper describes a new approach to implement Dynamically Allocated Multi-Queue (DAMQ) switching elements using a technique called "self-compacting buffers". This technique is efficient in that the amount of hardware required to manage the buffers is relatively small; it offers high performance since it is an implementation of a DAMQ. The first part of this paper describes the self-compacting buffer architecture in detail, and compares it against a competing DAMQ switch design. The second part presents extensive simulation results comparing the performance of a self-compacting buffer switch against an ideal switch including several examples of k-ary n-cubes and delta networks. In addition, simulation results show how the performance of an entire network can be quickly and accurately approximated by simulating just a single switching element.

1 Introduction

An n by m buffered switch is a critical component in many multiprocessor interconnection networks, including k-ary n-cubes and multistage Delta networks. The performance of these networks is closely related to the architecture of the switch from which it is constructed. This paper describes the design and evaluation of a new switch architecture using "self-compacting buffers".

The "self-compacting buffer" technique is an efficient approach to build a dynamically allocated multiqueue (DAMQ), which is one member of a family of switch architectures defined by Tamir and Frazier [1] (Figure 1). With FIFO switch (Figure 1(a)), packets in an input port may get blocked if they are not the first packet to be routed. The blocking problem can be resolved by providing separate FIFO queues for each output port at every input port. This scheme is called *statically allocated, fully connected* (SAFC) switch (Figure 1(b)). The problems with SAFC are that it requires expensive hardware resources and the utilization of buffer space at the input ports are not as good as FIFO switches. The amount of hardware used in the SAFC can be reduced by using one crossbar switch as shown Figure 1(c), which is called *statically allocated multi-queue* (SAMQ). Yet, the buffer space is still inefficiently utilized. A better way of using the buffer is to dynamically allocate buffer space in a SAMQ. Figure 1(d) shows this scheme which is called *dynamically allocated multi-queue* (DAMQ). It is reported that the DAMQ switch achieves the best performance among four switch types [1]. Finally, it is useful to consider the *ideal switch*, which provides a useful reference point for evaluating the other switch types. In the ideal switch, a multi-ported buffer is associated with each output port. If multiple packets request a given output port in the same cycle, the output port can accept all packets simultaneously. This implies that there is no blocking at input ports, unlike the other switch organizations mentioned above. Because of its high complexity, an ideal switch would be very difficult to implement in practice.

This study considers two aspects of DAMQ network switch design. The first is a new approach to implement DAMQ switches with virtual cut-through [2] flow control. The second is an investigation of DAMQ switch performance for a broad class of multiprocessor networks. We focus on DAMQ switches because they offer the best performance of the various switch architectures examined by Tamir and Frazier [1]. We also focus on virtual cut-through because it is commonly used and offers better performance than alternative flow control schemes. We extend published performance studies in two ways. First, we consider a much

1063-9535/94 $4.00 © 1994 IEEE

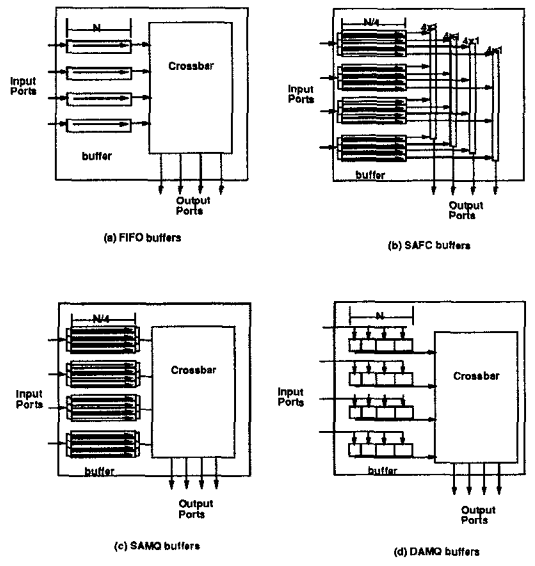

Figure 1: Alternative designs of switches with input port buffers

broader class of multiprocessor networks, including k-ary n-cubes and Delta networks. Second, we introduce the ***single switch simulation*** technique, which permits the performance of large networks to be accurately approximated with considerably less simulation time. The main contributions of this work are: (1) The design of an efficient way to build DAMQ switches using self-compacting buffers. This design offers comparable performance to a previously published design (the UCLA *ComCobb* switch [1]) at lower hardware cost. (2) Extensive simulation results comparing the performance of a self-compacting DAMQ switch against ideal and FIFO switches. The comparison extends previous work by considering a much broader range of network topologies, including several examples of k-ary n-cubes and delta networks. (3) Introduction of the "single switch" simulation method. This technique uses the simulation of a single switching element to approximate the performance of an entire network. It drastically reduces simulation time and the complexity of the simulator program itself, with little effect on accuracy. Single switch simulation is applicable to buffered networks in which the channel utilizations and routing probabilities are identical (or almost identical) from switch-to-switch.

The remainder of this paper has been organized as follows. Section 2 presents the design of a DAMQ switch using the self-compacting buffer technique. The performance of DAMQ switches in k-ary n-cubes

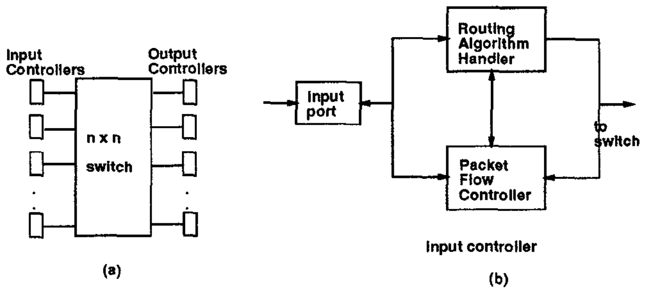

Figure 2: (a) Logical Blocks of a Router. (b) Logical Blocks of a Input Controller.

and Delta networks is examined in Section 3. The final section summarizes the main conclusions of this work.

2 Implementing DAMQ Switches with Self-Compacting Buffers

Logically, the router can be viewed as being composed of the input controllers, the (n by n) switch and the output controllers. The input controller receives incoming packets, performs the routing algorithm for the packet and determines the appropriate output channel number. The (n by n) switch delivers the packets from n input controllers to the n output controller and the output controller sends the packet to the neighboring node. Figure 2 (a) and (b) show an example of block diagram for a router and the input controller. The function of the input controller can be viewed from three perspectives. First, the input controller is responsible for receiving the packet and distributing the header part of the packet to the routing algorithm handler and to the packet flow controller. Second, determine the output channel number based on the header information which is received from the input controller. This task is carried out by the routing algorithm handler. Third, allocate and deallocate the buffer space for incoming and outgoing packets. In this section, we present a packet flow controller architecture that implements the DAMQ switch with a self-compacting buffer.

2.1 Self-Compacting Buffers

The packet flow controller consists of a buffer, buffer controller, channel pointers, the case selector, a new header register, an output channel number reg-

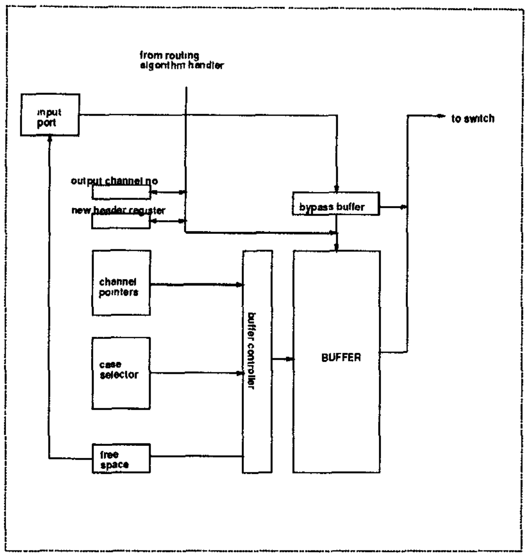

Figure 3: Logical Structure of Packet Flow Controller

ister, a free space register and a bypass buffer. The logical structure of the packet flow controller is shown in Figure 3. A detailed description of the packet flow controller components follows.

Buffer Management Scheme: The self-compacting buffer is divided dynamically into regions with every region containing the data associated with a single output channel. This scheme supports the DAMQ buffer management method introduced in [1]. The self compacting buffer scheme has the following properties:

Property 1: If two channels are denoted as i, k with $i < k$, then the dynamically allocated region for channel i and k always resides in a space addressed by addresses A_i and A_k respectively where $A_i < A_k$.

Property 2: There is no reserved space dedicated for a channel i. If no data are currently requiring the output channel i, then there is no region reserved for channel i.

Property 3: Within the space for each channel, the data are stored in a FIFO manner.

Property 4: For every output channel i, there is an integer number, δ_i, denoting the number of entries present in the region reserved for output channel i.

The properties of the buffer organization suggests that when an insertion/deletion in the buffer occurs via a write/read operation, there should be a mechanism to access arbitrarily the region that is associated with a channel. In particular, if the insertion of the packet requires space somewhere in the middle of the buffer, the required space must be created by moving

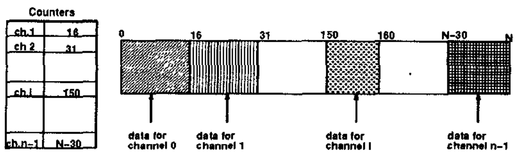

Figure 4: Buffer Space

all the data which reside below the insertion address. Furthermore, the reading from the top of the region for output channel data may create empty spaces in the middle of the buffer. The data below the read address need to be shifted up to fill the empty spaces. In the section to follow, we discuss in detail a high performance self compacting capability. The buffer space maintained under the self compacting buffer scheme is shown in Figure 4.

Buffer Organization: The buffer consists of n storage locations. Each storage location can load and store data. For a storage location i, the following actions can occur.

- shift up: storage location i can transfer its content to storage location $i - 1$,
- shift down: storage location i can push down its content to storage location $i + 1$,
- no action: storage location holds data.

Each storage location has a tag and a data field associated with it as shown in Figure 5. The tag field specifies the types of actions of a storage location. The data field simply stores the data. The "u" (shifting up), "d" (shifting down) and "e" (end of packet) are three bits in the tag field. When a request comes in to read/write data from/into a region, each storage location takes an action according to these three possible tags.

Buffer Operations and Case Selector: The buffer can read and write simultaneously. Depending on read, write or read/write operations(done in parallel), the tag bits in all storage locations have to be determined accordingly. There are four distinct cases by which the actions of each storage location in the buffer are determined. The function of the case selector is to determine the type of data movement and feed this formation to the buffer controller. Four cases of data movement are explained next.

case 1). Single Write (Insertion): For a given address to write data in, all storage locations whose ad-

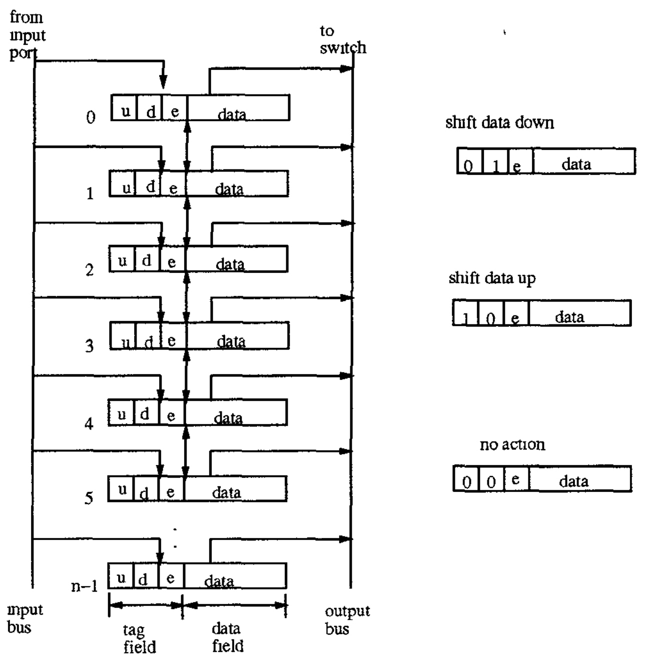

Figure 5: Buffer Organization

dresses are less than the write address leave their data untouched. The storage locations whose addresses are greater than or equal to the write address shift their contents down to open a space in the buffer for incoming data.

case 2). Single Read (Deletion): All storage locations whose addresses are less than the reading address leave their data as they are. The rest of the storage locations shift the contents of their storage location up.

case 3). Simultaneous Read and Write (address of read < address of write): In this case, the storage locations with addresses smaller than the read address are not affected. The storage location with addresses which are greater than the read and less than or equal to the write address should shift their contents upward. The rest of the storage location take no action.

case 4). Simultaneous Read and Write (address of write < address of read): In this case, only the storage location whose addresses are greater than or equal to the write address and less than the read address, shift their contents upward and the rest of the storage location require no action.

Buffer Controller: The buffer controller manages the tag of all storage locations in the buffer and it con-

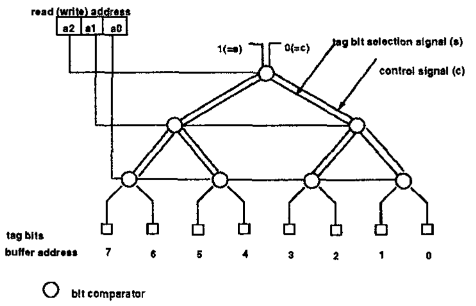

Figure 6: Logical View of Binary Tree Method.

trols the read and write operations. The inputs of the buffer controller are the case number from the case selector, read address and the write address. Once the buffer controller receives all the inputs, it determines the correct bit settings and sets the first three bits ("u","d" and "e") in the tag for all storage locations. The tags in all the storage locations are set in parallel. Supporting parallel tag bit settings can be done by associating a comparator to each of the storage locations. Then, the address of the buffer and the read(write) address are fed into the comparator to decide whether the shifting up or down bit (no action if both bits are 0s) should be set to 1 or 0. This scheme results in fast decision making. However, it requires n comparators(with two inputs of $\log_2 n$ bits) for the buffer of size n. We propose parallel bit settings that can be achieved using $n-1$ bit comparators(with two inputs of 3 bits each) in $\log_2 n$ time. This method uses comparators organized in binary tree fashion with one control signal(c), one tag selection signal(s) and one address bit as shown in Figure 6. The basic idea of this method is to divide a buffer address into two spaces and set the tag bit in one space to 0 and the tag bit in the other space to 1. For given buffer address starting from 0 to $n-1$, the interval of one space will include from 0 to i and the interval of the other space from $i+1$ to $n-1$. The buffer address increases from right to left in the tree. For a buffer with size n, its address can be represented by $a_{p-1}a_{p-2}...a_0$ where $p = log_2 n$. The left most bit of the address is fed into the comparator at the top of the tree and the second left most bit of the address to comparator at the second level of tree and so on. In addition to the address bit, the tag bit selection signal (s) and the control signal (c) are used as inputs to the comparator. The initial values for the signals "s" and "c" are 1 and 0. The "s" signal carries bit setting information. It will be 0

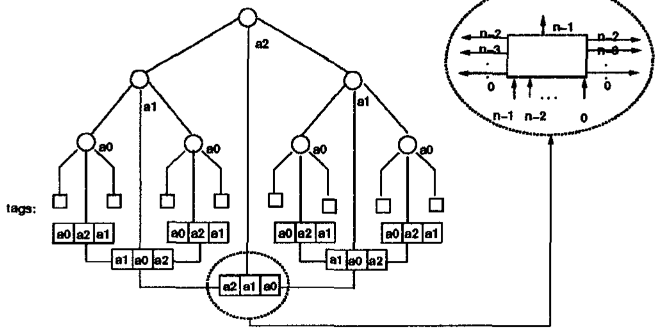
Figure 7: Physical Organization of How Address Bits Are Input to Comparators.

or 1 when it reaches the leaf node. The "c" signal is used as control signal. Whenever the comparator at the node receives the "c" signal with value 1, it means that the decision for the node and the subtree of the node are determined. Signal "s" is then propagated to its children. Figure 7 shows how address bits are fed into the comparator tree. Address feeding logic is another tree whose number of nodes is equal to the number of nodes in the bit setting controller. The function of each node in the address feeding logic is the same. Each node sends the most significant bit of its content to the bit setting controller. Then it rotates its content one position to the left, and sends the content to its left and right child.

Bypass Buffer: The bypass buffer is an intermediate storage between the input port and the buffer. There are two cases where the bypass buffer is used. Case 1). When the input port starts receiving data from the paired output port, the data have to be held until the routing algorithm handler determines the output channel number. The bypass buffer is used as an intermediary storage so that the input port can receive the incoming data while the routing algorithm handler executes the routing algorithm.
Case 2). The bypass buffer is also used as the flit buffer [3, 4]. In this case, cut-through occurs and data are forwarded to crossbar switch directly from the bypass buffer.
Channel Pointers: There is a channel pointer for each of the output channel. A channel pointer for channel i points to the beginning address of data queued for channel i. An example organization of the packet flow controller is shown in Figure 8.

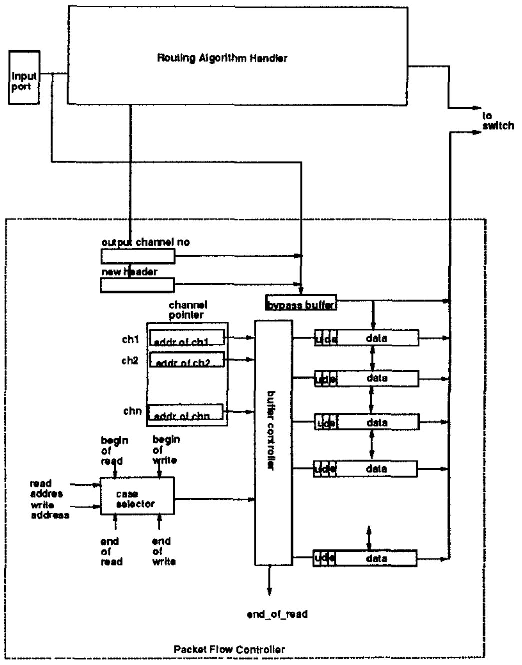

Figure 8: An Example Organization of Packet Flow Controller

2.2 Timing

The ComCobb chip from UCLA is the first switch that implemented the DAMQ [1]. The DAMQ scheme requires complex buffer management operation. The ComCobb chip used linked list concepts to dynamically allocate the buffer spaces. In the self-compacting buffer, the buffer controller with the case selector and the channel pointers are two key components for the buffer management operation. To minimize the overhead of the buffer management operation, the buffer management operation is overlapped with data transmission/reception. This is done by performing the buffer management operation for $(n + 1)th$ block of data while the $(n)th$ block of data is being received/transmitted. Thus, the time (δ) for the buffer management operation will not be seen if (δ) is less than the transmission/reception time of a block of data. This is shown Figure 9. Both the self-compacting buffer scheme and the ComCobb chip utilizes the same concept of overlapping the buffer management with data transmission/reception and achieves the same performance with respect to timing.

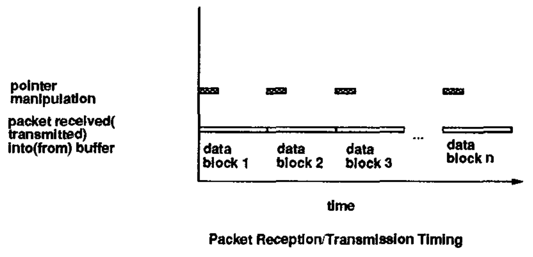
pointer manipulation
packet received(transmitted) into(from) buffer
data block 1
data block 2
data block 3
data block n
time
Packet Reception/Transmission Timing

Figure 9: Timing Diagram of ComCoBB Chip

2.3 Complexity

In the ComCoBB chip, each block of the buffer is associated with a header byte register, a length byte register and a pointer register. Those registers are necessary to implement linked list to support the DAMQ buffer management. The ComCoBB chip has four channels and each channel can have maximum of four blocks. In the self-compacting buffer, each block of the buffer has a tag and a shifter. There are $(n+1)$ channel pointers for n channels. Table 1 shows comparison of hardware complexity between the ComCobb chip and the self-compacting buffer. From this table, we can derive the overhead function as following:

$$overhead(ComCobb) = \frac{(ln(n) * (n+10) + n * 18)}{(n * t * 8)}$$

and

$$overhead(self - compact) = \frac{(ln(n) * (n+4) + n * 3)}{(n * t * 8)}$$

The self-compacting buffer has significantly less overhead with respect to latches than the ComCobb chip. The table 2 shows the calculation of overhead with several different buffer and block sizes.

Table 1: Hardware complexity of ComCobb chip and self-compaction buffer

ComCobb chip		
Function	*size*	Quantity
pointer	ln(n)	n
header	8 bits	n
length	2 bits	n
head	ln(n)	5
tail	ln(n)	5
data	t bytes	n
shifter	4 bits	2*n

Self-compaction buffer		
Function	*size*	Quantity
tag	3 bits	n
chan ptr	ln(n)	5
data	t bytes	n
shifter	ln(n)	n-1

n = the number of blocks
t = the number of bits per block

Table 2: An example overhead calculation with 8 bytes per block.

buffer size	*ComCobb*	*Self-comp.*
2	37 5%	9 37%
4	39.1%	10 8%
8	38.7%	9.4%
16	38.3%	12 5%

3 Performance of DAMQ Switches in k-ary n-cubes and Delta Networks

This section compares the performance of DAMQ, FIFO and ideal switches. The data show how closely a realistic switch design (the DAMQ switch) approximates an ideal switch, which is much more expensive to implement. The data expand the data reported by Tamir and Frazir [1] by considering a broader class of network topologies, and multiple packet sizes. Tamir and Frazir's results considered only a 64 node Omega network and single flit packets. This section reports results for several examples of k-ary n-cubes and Delta networks for multiple, fixed packet sizes.

This section also presents a useful technique for making fast, accurate approximations of network performance using results from simulations of a single switch.

3.1 Methodology

The principle metric for comparing the different switch implementations is the average latency experienced by a packet traveling through a network constructed from a collection of such switches connected in some topology. The topologies considered are specific examples of k-ary n-cubes and Delta networks. The data reported in this section comes from three sources:

1.*Published Data*: All of the data for ideal switches come from published simulation results used to validate various analytic models [1, 5, 6].

2.*Simulations of Complete Interconnection Networks*: These were obtained from a network simulator instrumented to collect statistics such as channel utilization, latency and routing distribution of DAMQ switch.

3.*Simulations of Single Switches*: For k-ary n-cubes and Delta networks, each switch in the network has the same set of routing probabilities from inputs to outputs, and the same set of channel utilizations on the input and output ports. This symmetry can be exploited to approximate the performance of a complete network with simulation results for a single switch. This approximation has been used in several analytic models of k-ary n-cubes [5, 6] and Delta networks [7], assuming ideal switches. The data presented here applies to DAMQ switches. We present data for single switch simulations to show that this approximations compares very favorably with the data from full network simulations.

All simulations were made with the following assumptions:
1. Infinite buffers.
2. Uniform packet destinations.
3. Fixed packet size.
4. Packet interarrival times are geometrically distributed with parameter p (ie. the probability that the next packet arrive after n dead cycles is p). This implies an average arrival rate of p.
5. Infinite buffers at the source and destination "processors".
6. Virtual cut-through flow control.
7. The latency of a packet transmission was measured from the time the first flit of the packet is injected into a network switch to the time that the last flit leaves th 'ast switch in its path.
8. Steady-state was assumed to be reached by simulating a large number of network cycles.

3.2 Performance of k-ary n-cubes Constructed with DAMQ Switches

A k-ary n-cube is a network with n dimensions having k nodes in each dimension. The k-ary n-cube network has the same set of channel utilization on the input and output ports. Figure 10 shows an example of routing probability of each input and output channel for 1024 node 3D Torus network. The k-ary n-cube network uses the dimension ordered routing on virtual channel developed by Dally [3]. In our simulation, the unidirectional channels were assumed for simplicity.

Under a uniform workload and dimension ordered routing on virtual channels, each switch in a k-ary n-cube network has the following properties:

1. The channel utilization of the n routing channels is given by:

$$\rho = \frac{mBnk_d}{n} = mBk_d$$

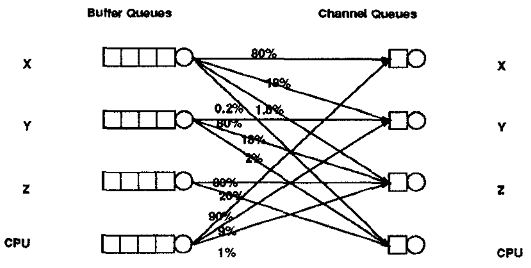

Figure 10: Routing Probability Distribution for a Unidirectional 3-D Torus Network.

where ρ is channel utilization, m is message generation rate, n is network dimension, B is message size, k_d is the average distance a message must travel in each dimension [5].

2. The channel utilization of the input and channels of the local processor is m.

3. The routing probabilities, $(R_{i,j})$, for given n input channel i and output channel j are:

$$R_{i,j} = (k-2)/(n) \qquad \text{for } i = j$$

and

$$R_{i,j} = (2)/(k) * k^{-(n-j)} \qquad \text{for } i = n+1$$

and

$$R_{i,j} = (2)/(k) * k^{-(i-j-1)*(1-1/k)}$$

for the rest of channels.

4. The routing probabilities for the input port i from the local processor to an output channel j are:

$$R_{i,j} = k^{-(n-1)} \qquad \text{for } j = n$$

and

$$R_{i,j} = k^{-(j-1)} * (1 - 1/k)$$

for the rest of channels.

A simulation run of a single switch provides the average waiting time, w, per packet. Then, the average latency of a message, T, through the network can be calculated by:

$$T = (1 + wB)nk_d + B.$$

for networks that use virtual cut-through. Here, $(1 + wB)$ represents the delay at a switch and multiplying the average distance(nk_d) to it, we can get

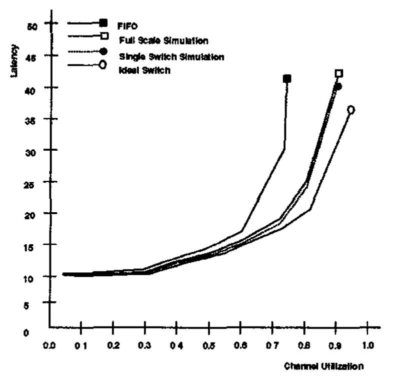

Figure 11: Comparing the network latency of DAMQ scheme to ideal and FIFO buffer management scheme for 3-D Torus ($k = 8$ $n = 3$).

the average latency for a unit packet. Since, we are assuming that the virtual cut-through is used, B is added to get the average latency for a message.

3.2.1 Fixed, Unit-Length Packets

Figures 11 to 13 show plots of average latency versus channel utilization for the following k-ary n-cubes, assuming fixed, unit-length packets:
1. $n = 3$, $k = 8$ (Three dimensional mesh with end-around connections.)
2. $n = 2$, $k = 10$ (Two dimensional mesh with end-around connections.)
3. $k = 2$, $n = 8$ (Hypercube.)

Each plot compares ideal, FIFO and DAMQ switches. Single switch DAMQ results are also shown to validate the single switch approximation. The data for ideal Latency for ideal switches were taken from [5] and [6]. Channel utilization was measured at the class of channel with the highest amount of traffic (the "bottleneck" channels). For the mesh cases this is any of the n routing channels. For the hypercube these are the "processor" input channels.

As expected, DAMQ switches provide significant improvement over FIFO switches, and do not perform quite as well as ideal switches. The single switch results provide a very good approximation of full-network performance. Furthermore, the single switch simulations took 10 to 100 times less simulation time than the full network simulations. Although the differences are small, the single switch model consistently underestimates the latency for high channel utilization. The reason for this was described by Agarwal

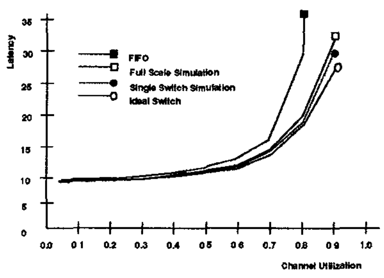

Figure 12: Comparing the network latency of DAMQ scheme to ideal and FIFO buffer management scheme for 2-D Mesh ($k = 10$ $n = 2$).

[5] for a similar effect encountered when extending single switch analytic results: in k-ary n-cubes using the dimension ordered routing algorithm, packets suffer higher-than-average delays in the higher dimension and was verified by simulation [5].

3.2.2 Fixed, Multi-flit Packets

Figure 14 shows the impact of increasing the packet size to 8 flits for an 8-ary 2-cube. The trends are similar to those observed for a packet size of one. The absolute latency values, however, increase markedly as the packet size is increased; the increase is roughly proportional to the increase in packet size. This corresponds to the fact that whenever a packet must wait in a queue, the wait time is proportional to the size of the packets in front of it. As before, single switch simulation provides a very good approximation.

3.3 Performance of Delta Networks Constructed with DAMQ Switches

A Delta network is defined as an a^n-by-b^n switching network with n stages consisting of a-by-b crossbar switches. In the Delta network, a simple *digit routing* is used. In the digit routing, a digit of routing tag at each stage determines which output channel should be used for routing data. The digit routing is inherently deadlock-free. Since Delta networks are multistage, no special ports are needed for local "processors".

An Omega is a particular example of a Delta network. A switch in a Delta network comprised of n-by-n switches, under uniform workload, has the following properties [7]:
1. The channel utilization of the n input and output channels is simply:

$$\rho = mB$$

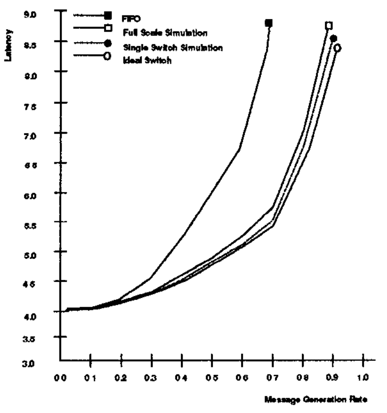

Figure 13: Comparing the network latency of DAMQ scheme to ideal and FIFO buffer management scheme for Hypercube ($k = 2$ $n = 8$).

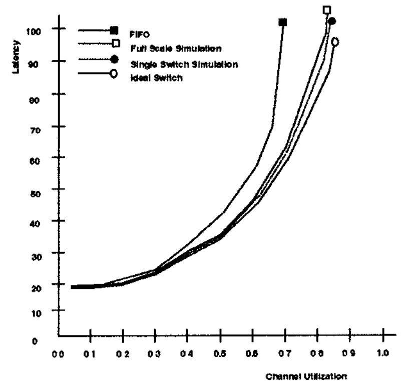

Figure 14: Comparing the network latency of DAMQ scheme to ideal and FIFO buffer management scheme for 2D Mesh ($k = 10$ $n = 2$). Packet size is 8 unit packets.

where m is message generation rate and B length of a message.

2. The routing probabilities for the n inputs are: $1/n$.

As for k-ary n-cubes, a simulation run of a single switch provides the average waiting time, w, per packet. The average latency of a message, T, through a complete network can be estimated by:

$$T = (1 + wB)n + B.$$

where n is the number of stage message travel and B the length of message. Similar to the average latency for a message in k-ary n-cube, $(1 + wB)$ represents the delay at a switch and multiplying the number of stages to it gives the average latency for a unit packet. Since the virtual cut-through is used, adding B to the average latency for the unit packet yields the average latency for a message of length B.

3.3.1 Fixed, Unit-Length Packets

Figures 15 compares the performance of four stage radix 4 Omega network, assuming fixed, unit-length packets: The data for ideal Latency data for ideal switches was taken from [7].

Again, the trends are the same as for k-ary n-cubes: DAMQ switches provide significant improvement over FIFO switches, and the single switch results provide a very good approximation of full-network performance.

3.3.2 Fixed, Multi-Flit Packets

Figure 16 shows that increasing the packet size increases the latency by a proportional amount, as was observed for the k-ary n-cubes. The saturation points remain about the same.

4 Conclusion

This paper has presented an efficient way to implement high performance switching elements using "self-compacting buffers". This technique offers the high performance possible with a Dynamically Allocated Multi-Queue, and requires less hardware than the alternative scheme proposed by Tamir and Frazier [1].

The second part presented extensive simulation results comparing the performance of a self-compacting DAMQ switch against an ideal switch. The comparison extends previous work by considering a much broader range of network topologies, including several examples of k-ary n-cubes and Delta networks.

Additional simulation results showed how the performance of an entire network can be quickly and accurately approximated by simulating just a single switching element. The single switch simulator required 10 to 100 times less simulation time, and was about 10 times smaller, than the full network simulator. As a specific example, the data plotted in Figure 13 took about 3 days of simulation time for the full networks, whereas single switch simulation took several minutes.

There are several ways in which this work can be extended. First, the single switch simulation technique can be applied to other network topologies and switch organizations in which the routing probabilities and

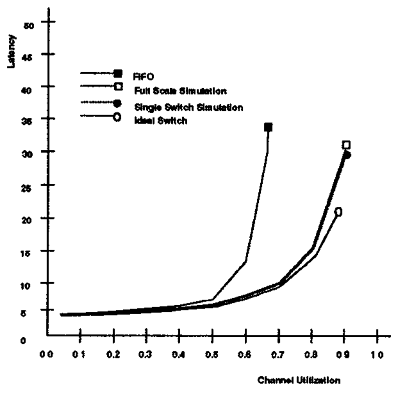

Figure 15: Comparing the network latency of DAMQ scheme to ideal and FIFO buffer management scheme for 4 x 4 switch Omega network with 256 nodes (4 stages).

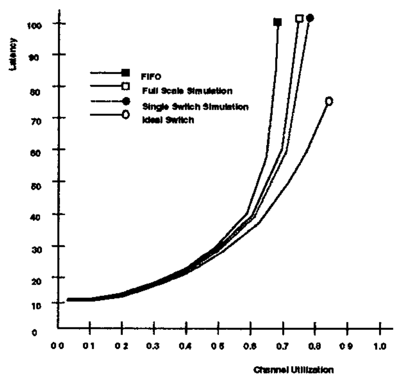

Figure 16: Comparing the network latency of DAMQ scheme to ideal and FIFO buffer management scheme for Omega network (256 nodes). Packet size is 8 unit packets.

channel utilizations are symmetric. Second, a real switch design would be pipelined over several clock cycles; it would be useful to do a performance study considering this effect. Third, it would be interesting to compare the performance of wormhole flow control versus virtual cut-through for a range of finite buffer sizes.

References

[1] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol.14, No. 6, pp. 725–737, 1992

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

[3] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. on Computers*, vol. C-36, No.5, pp. 547–553, May 1987.

[4] W. J. Dally, "Virtual-Channel Flow Control," *In Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 60–68, May 1990.

[5] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distribution*, vol.2, No.4, pp. 398–412, 1991.

[6] S. Abraham and K. Padmanahan, "Performance of the Direct Binary *n*-Cube Network for Multiprocessors," *IEEE Transactions on Computers*, vol.38, No.7, pp. 1000–1011, July 1991.

[7] C. P. Kruskal and M. Snir, "The Performance of Multistage Interconnection Networks for Multiprocessor," *IEEE Transactions on Computers*, vol.C-32, pp. 1091–1098, 1983.