Dynamic Buffer Organization Methods for Interconnection Network Switches

Amit Kumar Gupta, Francois Labonte, Paul Wang Lee, Alex Solomatnikov
{amit.gupta, flabonte, paulwlee, sols}@stanford.edu

I. INTRODUCTION

Interconnection networks originated from the design of high performance parallel computers. The need for more network bandwidth has put them to use in network switches and they may even be used to connect peripherals to a computer. The next step for interconnection networks is to be used as on-chip networks between different components on a chip. With scaling, transistor count and speed goes up, but the wires that span the chip don't get faster.

Traditional interconnection networks were limited by pin counts on chips. However, in on-chip networks wires can be packed so densely that this is no longer an issue. Instead, we would like the area of the switch to be small, to give more area for actual computation. Buffer space is a critical part of the area taken by a switch, and as such of critical interest for an on-chip network. Very few papers on on-chip networks have been written, so we will look into efforts of using buffers efficiently for general interconnection networks.

The function of the switch is to take packets arriving at its input ports and route them to its output ports. As long as only one packet at a time arrives for a given output port, there will be no conflict, and the packets are routed with minimum latency. Unfortunately, as the throughput goes up, so does the probability of conflict. When two packets destined for the same output port arrive at different ports of the switch at approximately the same time, they cannot both be forwarded immediately. Only one packet can be transmitted from an output port at a time, and hence one of the two packets is stored at the node for later transmission. The maximum throughput at which switch can operate depends directly on how efficient the switch at storing the conflicting packets and forwarding them when the appropriate output port is no longer busy.

An ideal switch has infinite buffer space, but will only buffer (delay) a packet as long as the output port to which the packet is destined is busy. Such a switch can handle *n* incoming packets while transmitting *n* packets and can receive and forward the first byte of a packet in a single cycle [8]. In a real implementation, buffers are finite and have a finite bandwidth. This can result in conflicts due to attempted simultaneous accesses to shared buffers and in messages that cannot be received due to lack of buffer space. Packets ready to be transmitted may be blocked behinds packets waiting for their output port to free up, and significant delays may be introduced by complex memory allocation schemes required to handle variable size packets.

Flow control techniques such as store and forward and virtual cut through require the switch to possess buffers large enough to buffer the whole packet. In wormhole flow control, where credits are fed back in the system to the upstream node, for one packet to flow through undisturbed, one needs enough buffers for the roundtrip in the wire and the time at each of the switches to give back credits and receive them.

But wormhole flow control makes bad usage of resources by holding on to channels when it is blocked. Virtual channel flow control [9] improves on this by interleaving packets on a channel and allowing deadlock avoidance without limiting as much route diversity. A simple scheme allocates a buffer for each virtual channel, but again, if only one virtual channel is to be used at full bandwidth, we need buffers for the roundtrip and work at both ends.

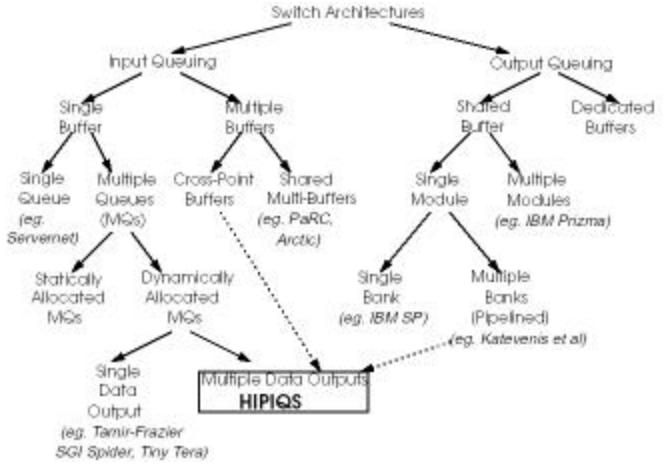

Fig. 1. Switch architectures buffer organization

Switch architectures can be compared by their organization of buffers as in figure 1 from [5]. Buffering at the output level is ideal because there is no longer any dependency on the input port, only on the destination (which is unavoidable). But buffering at the output is very complex and is rarely implemented. At the input level, one or many buffers can be allocated for an input port. Of interest to us are the ones with a single buffer that allow multiple queues to be dynamically allocated. The final distinction is between fully connected and partially connected. A fully connected switch allows an input port to talk to all data output ports at the same time.

The papers we have chosen to critique form a historical progression where each references all of the preceding ones. The 1992 paper by Y. Tamir and G. Frasier [1] talks first of a dynamically allocated multi-queue buffer. Then in 1998, R.

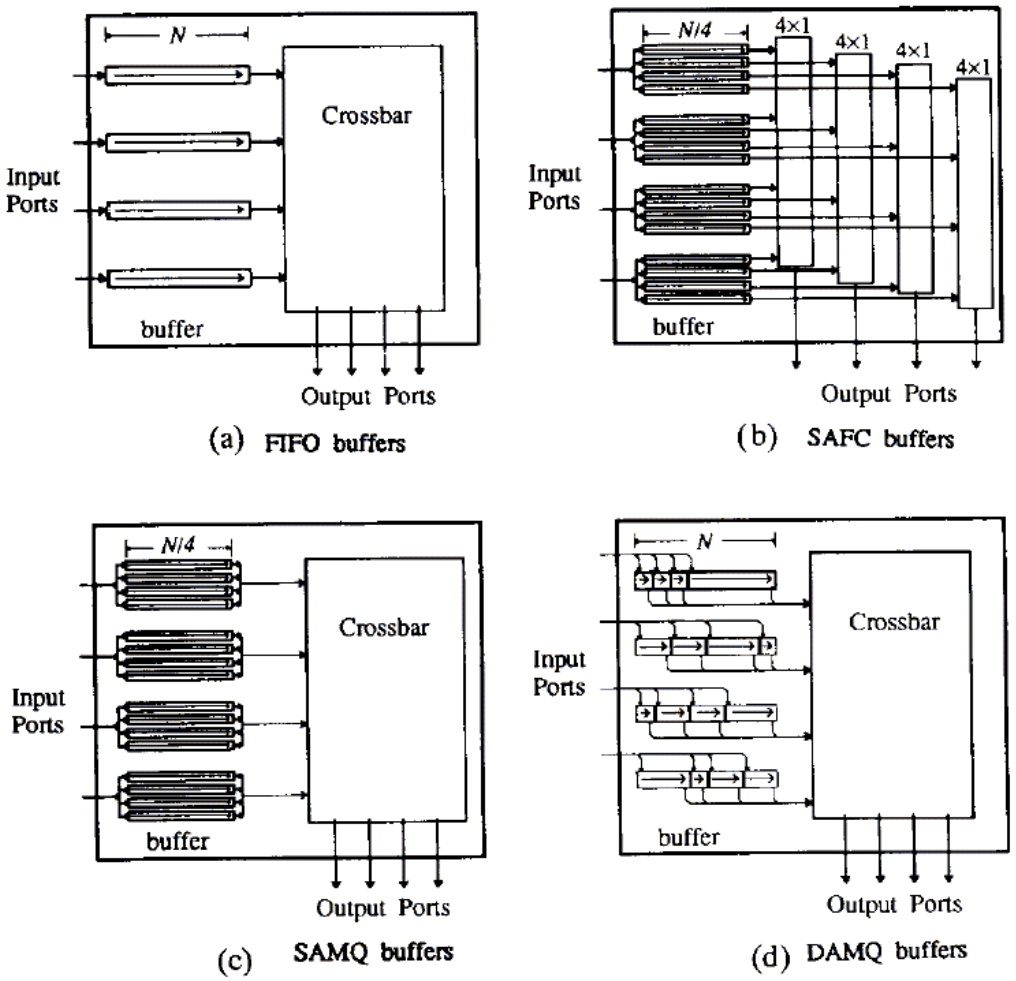

Fig. 2. Alternative designs of switches with input port buffers

Sivaram, C. Stunkel, D. Panka [5] add full connection to the scheme. Finally, in 1999 N. Ni, M. Pirvu and L. Bhuyan [6] adapt this method to wormhole routing with virtual channels and attempt to improve on the buffer allocation methods.

II. Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches

The paper "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches" [1] describes a newly proposed Dynamically Allocated Multi-Queue (DAMQ) buffering for communications switches. This research was a part of the UCLA ComCoBB (Communication Coprocessor Building Block) project, whose focus was a design and implementation of a single-chip high-performance communication coprocessor for a VLSI multicomputer [1], [2], [3], [4]. The problem the authors were trying to address was an efficient organization of internal buffers of communication switches. This organization is crucial for achieving high-throughput low-latency communication with cost-effective implementation.

The authors mainly considered input buffering schemes because unlike output buffering such schemes require a buffer with only a single write port. Four out of the five considered switch architectures are shown in Fig. 2.

The simplest architecture is a structure with FIFO queue input buffer shown in Fig. 2a. The advantage of this approach is simplicity: the buffer can be organized as a circular buffer with head and tail pointers. It is also easy to deal with variable length packets and avoid memory management problems such as fragmentation. However, the perfomance of such an architecture suffers because of the head of line (HOL) blocking problem: packets may be blocked unnecessarily in the input buffer if a single packet at the head of the queue whose destination output port is busy can block all other packets in that queue from being transmitted even if their destination output port are idle.

Another considered architecture is the so-called *statically allocated, fully connected* (SAFC) switch shown in Fig. 2b. The name stems from the fact that input buffers are statically partitioned into queues for different output ports. SAFC buffers have several problems. Complexity is increased because each input port requires several separate buffers and buffer controllers. The utilization of the available storage cells

is not as good as in the FIFO switch because available buffer space is statically partitioned so that only a fraction of the input buffer space is available as potential storage for any given packet in contrast to the FIFO switch. Another problem is the complexity of efficiently implementing flow control with blocking switches: with several separate buffers at each input port, information about each of these buffers must be conveyed to corresponding output port.

The next architecture considered is called the *statically allocated multi-queue* (SAMQ) switch (Fig. 2c). The space for the output ports is allocated statically, but it is implemented as multiple queues within a single buffer. Since the crossbar is the same as in the FIFO switch, some overhead of the SAFC switch is eliminated. However, the problems caused by the need to preroute packets and the inefficiency of static partitioning of the available buffer storage are the same as in the SAFC switch.

The switch architecture shown in Fig. 2d is the original contribution of this paper. It is called the *dynamically allocated multi-queue* (DAMQ) buffer. Dynamically allocated, because the space within the buffer is not statically partitioned between output ports, but is allocated on the basis of each packet received. Multi-queue because within each buffer there are separate FIFO queues of packets destined for each output port. Multiple queues of packets are maintained within a DAMQ buffer in linked lists. In order to manage linked lists of variable size packets, the buffer is partitioned into 8 byte blocks. Each packet occupies from one to four blocks. For each buffer block there is a pointer register, which points to the next block in the list. The pointers for the linked lists (the head and the tail) are stored in the separate storage arrays. There are five linked lists in each buffer: a list of packets destined for each of three output ports with which the input port is not paired, a list of packets destined for the processor interface, and a list of free (currently unused) buffer blocks. The DAMQ architecture achieves better utilization of buffer space than either the SAFC or the SAMQ architecture at the expense of increased complexity of linked list management logic.

Finally, the authors consider the so-called *centrally buffered dynamically allocated* (CBDA) switch. In such architecture a single buffer is used to store packets arriving from all input ports and destined to any to any output port. Complete sharing of available storage by all communication ports results in more efficient storage utilization than static partitioning of the buffer between input or output ports. Unfortunately, there are fundamental difficulties in producing efficient high-performance implementations of a switch with shared central buffers. In order to achieve high performance, multiple high-bandwidth communication ports must be able to access the central buffer simultaneously. Since multiport memory is undesirable because of area and long access times, such buffers are usually implemented as wide memory banks. However, the need to "assemble" bytes into wide words before transmission or storage increases communication latency especially in lightly loaded networks.

To compare performance of different switch architectures the authors used two different methods. First, they evaluated the performance for discarding switches, which discard packets that attempt to enter a full buffer, using Markov models. Although the ComCoBB project uses 4x4 switches, the authors calculated performance of 2x2 switches since the number of states for 4x4 switches is too large. They also made a number of simplifying assumptions: 1) fixed length packets, 2) uniform distribution of packets destinations, 3) when there is contention for an output port, the input port that is allowed to transmit is chosen randomly, and 4) synchronous store-and-forward operation of the switch so that during each stage cycle packets either completely arrive or completely depart. The main performance metric is the percentage of discarded packets versus applied input traffic rate. The results show that a DAMQ switch with space for three packets per input buffer discards as few or fewer than the FIFO switch with space for up to six packets for all traffic rates. Furthermore, the DAMQ switch performs significantly better than the FIFO switch for high traffic rates. The DAMQ switch outperforms both the SAFC and SAMQ switches for almost all traffic rates and input buffer sizes except the case of very heavy traffic (0.99) when the SAFC switch has insignificant advantage. A CBDA switch shows the best performance for all cases.

Next, the authors used simulation of a 64x64 Omega network constructed from three stages of 4x4 switches to evaluate different architectures. They simulated both discarding and blocking (i.e. which block the transmitter from sending to a full buffer) switches. In this part they also assumed synchronized, store-and-forward transmissions, where packets are transmitted/received instantaneously once every stage cycle. Fixed length packets and uniform traffic distribution were assumed. Again, discarding DAMQ switches showed better performance than FIFO, SAFC or SAMQ switches for any input traffic rate and input buffer size per input port. The performance of blocking switches is measured in terms of average latency versus applied input traffic and input buffer size for input port. The DAMQ switch outperforms FIFO, SAFC and SAMQ switches for most combinations of input traffic rates and input buffer sizes. The only case when SAFC has better performance than the DAMQ switch is the case of heavy input traffic and large input buffer when full connectivity of the SAFC switch plays a more important role than the efficient buffer usage of the DAMQ switch.

Finally, the authors also simulated "hot-spot" traffic, where a particular ("hot") destination receives a higher percentage of the packets than any other destination. In this case none of five considered architectures has clear performance advantage.

In general, the quality of this paper is very good. All concepts are presented clearly and properties of all considered architectures were analyzed thoroughly. However, in our opinion several issues were not addressed:

1) The proposed DAMQ architecture has much more

complicated control logic to support dynamic linked lists. This means a DAMQ switch would have lower clock frequency than a simple FIFO switch. In [2] the authors claim that clock rate advantage is not significant (48 MHz for DAMQ vs. 53 MHz for FIFO in MOSIS 2 μm technology) because clock rate would be limited by the speed of links. Although this statement may have been true in the late 80s when the paper was written, it may be not correct now when even chip-to-chip links can have 1 GHz or higher clock frequency.

2) It is not clear whether the comparison between different architectures in terms of only input buffer size is valid. Different architectures have different complexity of control logic and, therefore they would have different areas even for the same input buffer size. The only approximate comparison they made in [2] is with simplest FIFO switch.

3) It is not easy to compare Markov model calculations for 2x2 switches with simulation results for 4x4 switches, especially in the case of FIFO switch where conflicts due to head of the line blocking have significantly higher probability in the case of 4x4 switches.

4) The proposed DAMQ switch is supposed to be better than CBDA in handling packets of variable size but all calculations and simulations are performed with the assumption of constant packet length. The effect of variable length packets is not investigated at all.

5) Although the target application is a multicomputer interconnection network in which the latency is one of the most important parameters, all performance comparisons were made with assumption of store-and-forward instantaneous one cycle packet transmission. Virtual cut through or wormhole routing seem to be better alternatives for such kind of networks, although these techniques are harder to simulate.

III. HIPIQS : High Performance Input-Queued Switch

The HIPIQS [5] switch architecture is an input-queued switch. The basic idea of the design of this switch is to allow the input buffers to behave like a buffer with multiple read ports.

The switch is organized with k-input modules (k being the number of inputs and outputs) connected to k-output buffers through multiplexors. The buffering and control logic is contained mostly within the input modules.

The input module contains a shared input buffer with k banks of memory, a $(k+1)\times k$ crossbar and k FIFOs (one for each output). There are bypass paths available everywhere to allow lower latency if the switch has no traffic on the required input and output links. The first bypass is provided for the input buffer itself - hence the need for $k+1$ inputs to the crossbar. A second bypass is provided for the FIFOs, which can be used if there are no flits queued for that particular output.

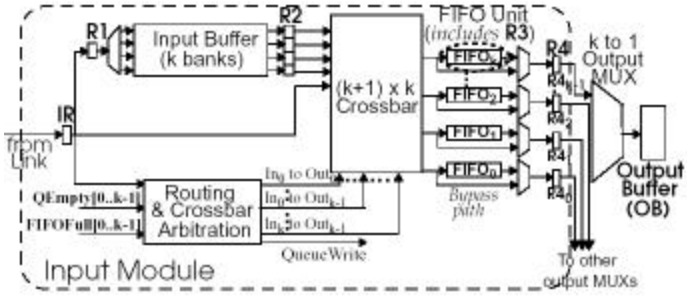

Fig. 3. HIPIQS Input Module

The multiple banks in the input buffer allow the transfer of packet flits to be pipelined. Packet transmission occurs in k-flit units known as chunks. The FIFOs should be sized to be able to accommodate at least 2 packet chunks, i.e. $2k$ flits.

The input buffer is managed using linked lists, 1 linked list for each output port, and 1 for the free list. The output queue lists are used as FIFOs and hence require storage of both the head and tail pointers. However the free list can simply be organized as a LIFO list, and hence only the head pointer needs to be stored.

An output port can begin reading packets from the buffer starting with the location pointed to by its head register. Flits from the banks are successively read into corresponding registers; this allows for pipelined transfer of the chunks, as flits are transferred across the crossbar while the next flit is being read from the input buffer. When the last flit of the chunk is read, the completely read chunk is added to the free list; and the next chunk begins to be read from the next location in the link list.

This architecture is designed for maximum efficiency when packet sizes are exact multiples of chunks. When a significant fraction of packets are smaller than a chunk, memory fragmentation can result in a significant loss of space, and there could also be a loss of bandwidth if multiple output ports want to read from the same input buffer more frequently than once every k cycles. This can be alleviated by allowing a chunk to contain multiple packets - but only those scheduled for the same output port.

When there is no contention for resources in the switch, it takes a minimum of 2 cycles for the flit to get from input port to output - one for the input module, and one for the output buffer. If the output queue is not empty, the flit needs to be buffered in the FIFO to prevent its overwriting the previous flit. This introduces an additional 1 or 2 cycles of delay depending on how the FIFO is implemented. If the FIFO is full, the flit needs to be sent to the input buffer. This introduces an additional 3 cycles of delay.

The authors compared the performance of this new switch architecture with 2 other architectures – a single-input queue (FIFO) model and a shared buffer output queue model, all having equivalent shared buffer space. The FIFO model is seen to perform considerably worse and hence discarded from further analysis. With varying message length, it is seen that for messages smaller than 20% of the input buffer size, HIPIQS performs almost as well as the output-queuing switch. For

messages that are larger than this fraction, output queuing benefits from better utilization of buffer space.

Increasing buffer size helps improve performance by raising the saturation throughput. Here again, once the packet size falls to about 20% of the buffer size, further increase in buffer size results in very little performance gain in terms of saturation load.

Simulation on larger systems (64-node instead of 16-node) show that HIPIQS performance scales well, but saturation throughput falls off faster than it does for output-queuing switches. Non-uniform traffic patterns (in particular bit-reversal and transpose) show saturation throughput numbers close to that of output-queuing switches.

Overall, this paper proposes a very interesting switch design that seems to perform well. But we think there are several concerns about this architecture:

1) HIPIQS architecture is very complicated. Essentially it performs buffering on both input and output ports. It also uses k $(k+1)\times$ k crossbar switches and complex control logic. All these factors complicate proper area and clock cycle comparisons of proposed architecture with others.

2) Arbitration/scheduling of proposed switch may be very complicated because there is buffering at both the input and the output.

3) It is not clear how the authors calculated the total buffer space used when comparing the various switch architectures. In particular, it is not clear whether the output FIFO buffers were included in their calculations of memory storage. This makes their comparisons with other architectures difficult or invalid.

4) The authors did not present any performance comparison with switch architectures of similar complexities, i.e. with a k^3 crossbar.

IV. Fully Connected Circular Buffer

Ni et al. [6] describe the FC-CB(Fully Connected Circular Buffer), a switch that adapts the DAMQ architecture to implement wormhole routing with virtual channels. It attempts to improve upon the basic DAMQ by using a new buffer management scheme using circular buffers, and by providing full connectivity for each virtual channel to every output port in a manner similar to the HIPIQS. The usage of shifting storage elements for the input buffers, in place of the linked list that was used in the original implementation of the DAMQ, was first described in Park et al. [7]. In this paper, the input buffer is maintained as a single shift-register-like structure, with each channel occupying a continuous stretch of buffer space, each located immediately adjacent to the next. The buffer is "self-compacting", in that all free space is at the beginning and the end of the buffer and every read and write operation on any channel results in appropriate shifts such that this property is preserved. Channel pointers point to the beginning of each channel to access the data. A comparison of hardware requirements with the original DAMQ implementation in the ComCoBB chip [1,2] is presented in the paper, and it is claimed that there is less hardware overhead compared to the ComCoBB implementation using linked lists.

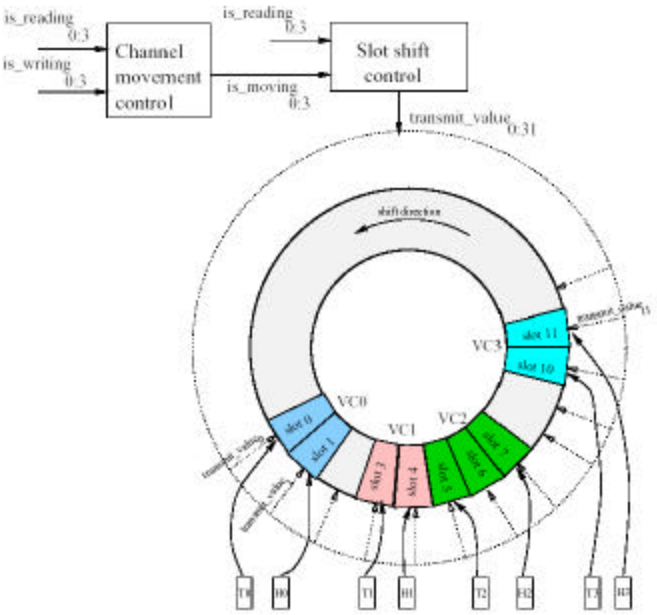

Fig. 4. Fully connected circular buffer

The circular buffer proposed by Ni et al. is a direct improvement of the self-compacting buffer [7]. The buffer is logically arranged as a circular structure with wrap-around. This eliminates the need to shift in both directions to manage the buffer, thereby reducing hardware complexity. The FC-CB also does not have to be compacted, due to the circular structure. Insert and readout can be done without affecting other channels except when an adjacent channel directly abuts the channel being operated upon, in which case a channel is "pushed" to make room. It is claimed that this enables simultaneous readouts from multiple channels.

In order to prevent a single virtual channel from consuming all buffer space, space for a flit is reserved for each virtual channel, resulting in minimum guaranteed progress.

Full connectivity is achieved using two alternatives. The first approach is to have an equal number of read ports per buffer as the number of virtual channels, and connect each of them to the bus, resulting in true full connectivity. This method is equivalent to the HIPIQS approach. The second approach is to have small buffers that are associated with each of the virtual channels, after a single readout bus. The authors claim that this is equivalent to providing full crossbar connection.

The paper presents simulation results to compare virtual cut through, wormhole, and wormhole with virtual channel flow control that the FC-CB implements.

The idea of using a circular buffer to simplify buffer management compared to the self-compacting buffer is a valid idea, and it does seem to reduce hardware requirements considerably, notwithstanding the weak justification for the

control logic as given in the paper. The unidirectional shift alone would result in large improvements both in complexity and area. It is not clear however, how such a shifting buffer would be constructed. In the linked list implementation of the first DAMQ, SRAM's may be used. It is conceivable to use a shift register as the shifting buffer, but this would imply a very much larger structure compared to an SRAM buffer. The argument that the FC-CB is superior to the linked-list implementation is based upon Park et al. [7]. However, the hardware comparisons in [7] do not seem to take into account the increased complexity and area of the shifting buffer. The buffer units also require that up to k arbitrary blocks be readable simultaneously where k is the number of virtual channels. It is unclear how this will be done, without incurring excessive penalty. A neglect that this paper shares with the HIPIQS paper is in the cost of the $k^2 \times k$ crossbar that a full connection requires. A full connection will always give better results, if the associated cost in area and timing constraints is ignored. A more careful analysis of the tradeoffs involved is required to properly assess whether the added ideal gain is indeed meaningful.

V. Overall Critique and Conclusion

Buffers are a limited resource in a switch and they need to be used efficiently to provide high-throughput low-latency communication. Dynamic allocation of input buffers presented in [1] allows efficient utilization of buffer space. [5] improves on this by providing a fully connected switch. [6] attempts to simplify the dynamic buffer management by use of a circular buffer.

Care must be taken in evaluating these switch architecture because there may be some tradeoffs that are not apparent. In terms of area, it is not fair to compare only buffer size because some buffers with additional operations (shifting) require more area. Buffer size comparisons do not have much meaning if they are not representative of the total area (including control logic, crossbars, etc). Clock frequency may also be a constraint with increased complexity.

Efficient buffer usage is even more important for on-chip interconnection networks, where the area devoted to switching is comparatively costly. An optimal solution for performance vs area is not self-evident and needs to be investigated further.

References

[1] Y. Tamir and G. L. Frasier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," in *IEEE Transactions on Computer*, 41(6):725-737, June 1996.

[2] G. L Frasier and Y. Tamir, "The Design and implementation of a multi-queue buffer for VLSI communication switches," in *Proc. Int. Conf. Comput. Deisgn*, Cambridge, MA, Oct. 1989, pp.466-471.

[3] Y. Tamir and G.L. Frazier, "High Performance multi-queue buffers for VLSI communication switches," in *Proc. 15th Annu. Int. Symp. Comput. Architecture*, Honolulu, HI, May 1988, pp.343-354.

[4] Y. Tamir and J.C. Cho, "Design and implementation of high-speed asynchronous communication ports for VLSI multicomputer nodes," in *Proc. Int. Symp. Circuits and Syst.*, Espoo, Finland, June 1988, pp. 805-809.

[5] R. Sivaram, C. B. Stunkel, and D. K. Panda, "HIPIQS: A High-Performance Switch Architecture using Input Queuing," in Proceedings of the IPPS/SPDP '98, Oregon, FL, Mar. 1998, pp. 134-143.

[6] N. Ni, M. Pirvu, and L. Bhuyan, "Circular Buffered Switch Design with Wormhole Routing and Virtual Channels", in Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. *Proc. Int. Conf. Comput. Design*, 1998, pp. 466 -473

[7] J. Park, B.W. O'Krafka, S. Vassiliadis, J. Delgado-Frias, "Design and Evaluation of a DAMQ Multiprocessor Network With Self-Compacting Buffers", in *Supercomputing* '94., Proceedings , 1994 pp. 713 –722

[8] P. Kermani and L. Kleinrock, "Virtual cut through: A new computer communication switching technique," in *Computer Networks*, vol. 3, no. 4, Sept. 1979, pp. 267-286.

[9] W. Dally, "Virtual-channel flow control," in *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194-205, 1992.