Proceedings of the
International Conference on Computer Design,
Cambridge, Massachusetts, pp. 233-238, October 1991.

Decomposed Arbiters for Large Crossbars with Multi-Queue Input Buffers

Hsin-Chou Chi and Yuval Tamir
Computer Science Department
University of California
Los Angeles, California 90024

Abstract

Crossbars are key components of communication switches used to construct multiprocessor interconnection networks. For a fixed number of nodes, larger crossbars result in reduced probability of conflicts and allows packets to traverse the network in fewer hops. However, increasing the size of the crossbar also increases the delay of the arbiter used to resolve conflicting requests. The increased arbitration delay can lead to overall poor network performance. The impact of the increased arbitration delay can be mitigated by decomposing the arbitration process into multiple steps, such that some requests can be granted before the arbitration of the entire crossbar is complete. This paper deals with the design of such *decomposed arbiters* for large crossbars. The focus is on crossbars with multi-queue buffers at their inputs. Such buffers have been shown to provide significantly higher performance than conventional FIFO buffers.

I. Introduction

Crossbar switches are commonly used in the interconnection networks of multiprocessors and multicomputers. Small networks can be implemented as a single crossbar while large networks are composed of many small crossbars [2, 1, 8]. Theoretically, throughput is maximized and latency is minimized if the entire network is one large crossbar. In general, given a choice between a small number of large crossbars or a large number of small crossbars, larger crossbar switches help lower the probability of conflicts and allow packets to traverse the network in fewer hops. With advances in VLSI fabrication and packaging technology, larger crossbar are becoming practical (e.g. the Inmos C104 packet-routing switch has 32 inputs and 32 outputs). A key factor in determining the performance of crossbars for communication is the mechanism used to arbitrate conflicting requests.

Several packets arriving at the switch simultaneously, to different input ports may be destined to the same output port. However, only one packet at a time may be forwarded through each output port. Due to the resulting contention for output ports, packets may have to be buffered at the inputs of the crossbar while awaiting service. Recent communication switch designs utilize *multi-queue* buffers, which maximize performance by allowing packets at an input port to be processed in *non-FIFO* order [9, 6, 10, 3]. Packets at each input port, which are destined to different output ports, may be forwarded through the switch in any order. Hence, each input port contends for *multiple* output ports but needs only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. The arbitration task is thus *symmetrical* with respect to inputs and outputs [11]. Since the arbitration result for each port is dependent on the arbitration for other ports, the crossbar arbitration for switches with multi-queue buffers is more complicated than for switches with FIFO buffers. We have previously proposed two high-speed *symmetric crossbar arbiters*, which efficiently solve the arbitration problem for multi-queue buffers [11].

One of the major challenges with large crossbar design is the long arbitration delay. Since the arbitration delay grows with the crossbar size, the arbitration speed of a large crossbar can become a performance bottleneck. The design of symmetric arbiters for large crossbars is the topic of this paper. It will be shown that high performance can be achieved by decomposing the arbitration process so that the arbitration of a large crossbar is performed using an array of smaller arbiters. Furthermore, in a large crossbar it is not necessary for the number of queues at each input port to be equal to the number of output ports. A small number of queues reduces the cost of buffer management and is sufficient for achieving high performance.

The next section addresses the subject of designing the crossbar arbiters for multi-queue buffers. Section III describes how to decompose the arbitration for large crossbars. Simulation results are presented and the logic and circuit design are described. The impact of varying the number of queues per input buffer is discussed in Section IV.

II. Crossbar Arbiters for Multi-Queue Buffers

Crossbar switches often include FIFO buffers at their input ports for storing incoming packets that cannot be forwarded immediately due to output port contention or blocking [4]. Figure 1-a shows the organization of a crossbar switch with FIFO buffers. The strict FIFO order of handling packets at each input port unnecessarily reduces the throughput of the switch [10]. When the packet at the head of the queue is blocked, all other packets in the same buffer are also blocked, even if they are destined to idle output ports.

Multi-queue buffers avoid the shortcomings of FIFO buffers by partitioning each input buffer into several queues. The packets at the head of the queues in a buffer may be accessed in any order. Even if one of the queues of the buffer is blocked, it may be possible to transmit a packet from the head of another queue, which is destined to a different output

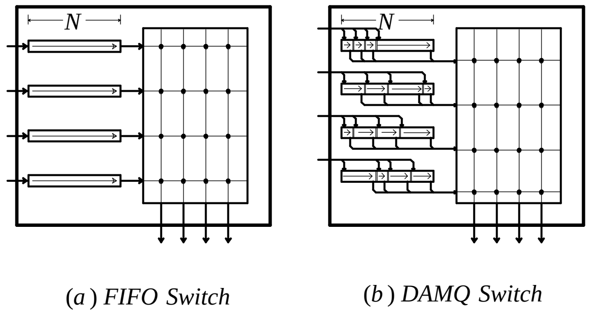

Figure 1: Switches with FIFO buffers and DAMQ buffers

port [9, 6, 10, 3]. A possible organization of a multi-queue buffer is to maintain one FIFO queue for each output port. In order to utilize buffer storage efficiently, it is desirable for all the queues to share common storage. A multi-queue buffer with this organization is called a *dynamically-allocated multi-queue* (DAMQ) buffer [10]. Figure 1-b shows the organization of a crossbar switch with DAMQ buffers. While the circuits introduced in this paper apply to most multi-queue buffers, the performance evaluation was done for DAMQ buffers.

Since multi-queue buffers may generate multiple requests (one for each queue) to the crossbar arbiter, there is an opportunity to connect more crosspoints than with FIFO buffers, thus leading to higher throughput and lower latency. Figure 2 shows an example of buffer contents and how requests can be arbitrated for FIFO and DAMQ buffers [11]. The numbers in the buffers indicate the destinations of the packets. The crosspoints with single circles represent denied requests, while those with double circles represent granted requests. The job of the arbiter for this type of multi-queue buffers is to arbitrate among up to n^2 requests in an $n \times n$ crossbar so that only one request is granted in a row and in a column.

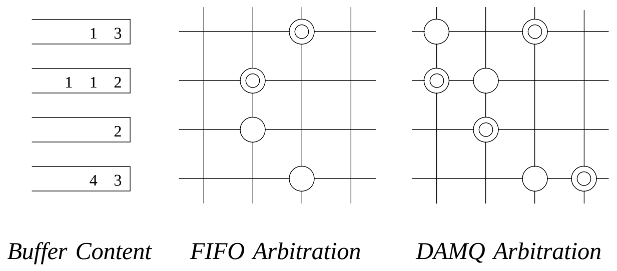

Figure 2: Example arbitrations for switches with FIFO and DAMQ buffers. Double circles indicate granted requests. Single circles indicate denied requests.

Figure 3 shows a 4×4 symmetric crossbar arbiter design, which is called a *wave front arbiter* (WFA) [11]. The arbiter is composed of n^2 arbitration cells, and each cell corresponds to a crosspoint. Double squares indicate that the corresponding crosspoint has been requested. Shaded squares indicate that the corresponding crosspoint has been granted. For each crosspoint (i, j), there is a *request* $(R_{i,j})$ input and a *grant* $(G_{i,j})$ output.

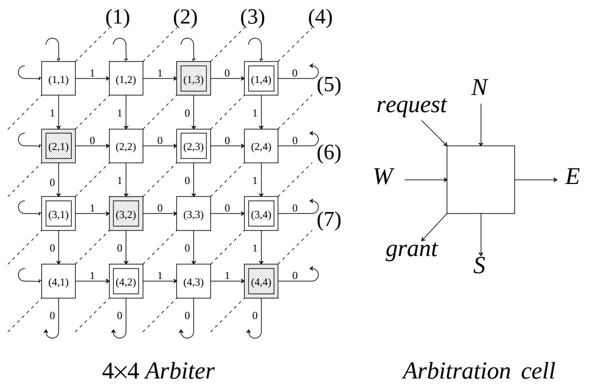

Figure 3: Wave front symmetric crossbar arbitration.

In addition, each cell has two inputs, *north* $(N_{i,j})$ and *west* $(W_{i,j})$, and two outputs, *south* $(S_{i,j})$ and *east* $(E_{i,j})$. Note that $N_{i,j} = S_{i-1,j}$ and $W_{i,j} = E_{i,j-1}$. The $N_{i,j}$ signal indicates that the rows above did not request column j. The $W_{i,j}$ signal indicates that there are no granted requests for the crosspoints to the left in the same row. The G output is asserted if, and only if, the crosspoint is requested and both the N and the W inputs are asserted. Thus, $G_{i,j} = R_{i,j} \wedge N_{i,j} \wedge W_{i,j}$, $S_{i,j} = N_{i,j} \wedge \overline{G_{i,j}}$, and $E_{i,j} = W_{i,j} \wedge \overline{G_{i,j}}$. For the highest priority row and column, all the N and W inputs, respectively, are set to 1.

The arbitration cell can be implemented as a simple combinational circuit [11]. Arbitration starts with one top priority cell and the arbitration cells reach their final configuration in a ‘‘wave front’’ that moves diagonally from the top left corner to the bottom right corner of the arbiter. In order to maintain fairness, the top priority is given to a different cell every cycle. Hence, every cell has the top priority every n^2 cycles. If the propagation delay for a cell is T time units, the whole arbitration completes after $(2n-1)T$ time units [11].

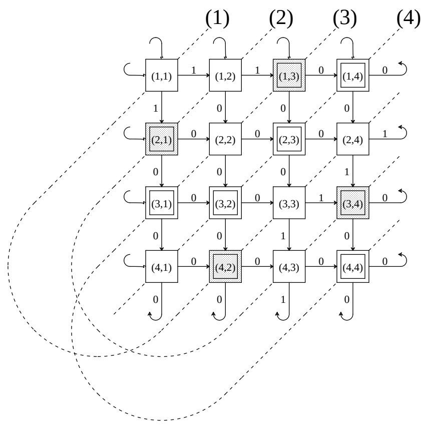

Figure 4: Wrapped wave front symmetric crossbar arbitration.

A similar symmetric crossbar arbiter, called a *wrapped wave front arbiter* (WWFA) is shown in Figure 4. The arbitration cells used in WWFA are exactly the same as in WFA. The arbitration wave front in the WWFA passes through n cells in each T time units. Hence, arbitration is faster than with WFA, completing in only nT time units. To rotate the priority, only one circular shift register is needed, which indicates the top priority n cells. Our analytic and simulation results show that WFA and WWFA achieve approximately the same network throughput and latency [11]. Since the WWFA is faster, it is the arbiter of choice, especially when the arbitration delay is critical.

III. Decomposition of Arbitration

For interconnecting a fixed number of nodes, larger crossbars lower the probability of conflicts and reduce the number of hops for a packet to traverse the network. Hence, theoretically, larger crossbars improve the performance of the interconnection network. However, the crossbar arbitration delay increases as the size of the crossbar increases. Thus, if large crossbars are used, arbitration delays can become a critical performance bottleneck. To study the speed of crossbar arbitration, a CMOS implementation of a 4×4 crossbar and its wave front arbiter have been laid out using the MOSIS scalable design rules [11]. With 2 μ technology, circuit simulations indicate that the worst case delay for reaching an arbitration result is 15.5 ns. For 2×2 and 8×8 crossbars, the worst arbitration delays are 7.2 ns and 32 ns, respectively. In general, for the WFA and WWFA arbiters, the arbitration delay grows linearly with the crossbar size.

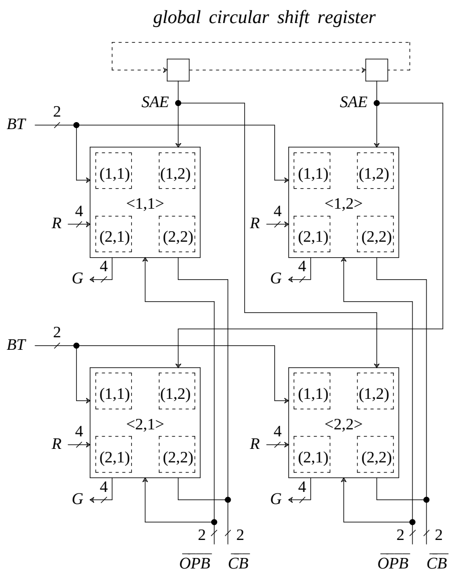

Figure 6: A 4×4 decomposed arbiter with 2×2 subarrays.

In principle, it is possible to construct arbiters whose delay increases logarithmically with the size of the crossbar [7]. However, these arbiters are much more complex than the WFA or WWFA and will therefore lead to increased implementation cost (chip area). Furthermore, in practice, due to the increased complexity, logarithmic arbiters are expected to result in lower performance for medium-size crossbars (e.g. 32×32). For much larger crossbars, use of logarithmic arbiters may be considered (more on this in Subsection III.A).

If the arbitration must complete in one clock cycle, long arbitration delays for large crossbars may force the use of a slower clock. In order to overcome this problem, the arbiter may be allowed several clock cycles to complete the arbitration. However, during the arbitration new requests are blocked. Hence, the multi-cycle arbitration will reduce throughput and increases latency. In order to improve the performance, the crossbar can be logically partitioned into subarrays, and each subarray is arbitrated in one cycle. Requests can be granted as soon as the arbitration of the relevant subarray is completed. This leads to a significant reduction in the average delay to granting a request. We call this scheme *decomposed arbitration*.

Figure 6 shows the organization of a decomposed arbiter for a 4×4 crossbar. In the figure the whole arbitration array is partitioned into four subarrays, where each subarray is a 2×2 WWFA. With the simple WFA and WWFA, conflicting requests are guaranteed to be on different ‘‘wave fronts’’ [11]. The same principle is the key to the operation of the decomposed arbiter. Subarrays which are arbitrated in parallel, in the same clock cycle, have no inputs or outputs in common. Hence, it is guaranteed that requests which are arbitrated in different subarrays during each cycle cannot conflict. In the arbiter shown in Figure 6, two subarrays can be arbitrated during each cycle. For example, subarrays <1,1> and <2,2> may be arbitrated in the first cycle and subarrays <1,2> and <2,1> can then be arbitrated in the second cycle. This allows requests to <1,1> and <2,2> to be granted in the first cycle so that transmission can begin without waiting for the second cycle. A local circular shift register, which shifts every two cycles, is used for priority rotation within each subarray. Subarray arbitration is controlled by a global circular shift register which shifts every cycle.

A. Performance Evaluation

Event-driven simulations were used to evaluate decomposed arbitration. Simulations were performed for a 16×16 and for a 32×32 crossbar using three arbitration schemes: (I) ‘‘*ideal*’’ single cycle arbitration using a WWFA, (II) ‘‘*nondecomposed*’’ multi-cycle arbitration using a WWFA where the number of cycles to complete the arbitration is proportional to the size of the crossbar, and (III) ‘‘*decomposed*’’ arbitration using 4×4 WWFA subarrays. We assume that a 4×4 WWFA completes an arbitration in one cycle. Hence, a subarray in the decomposed arbiter operates in one cycle. With nondecomposed arbitration, it takes four cycles to arbitrate a 16×16 crossbar and eight cycles to arbitrate a 32×32 crossbar. The links and buffers are byte-wide and every cycle one byte can be read from a buffer, written to a buffer, or transferred through a link.

The simulations are based on traffic with the following properties: 1) packet size is evenly distributed between 8 and 32 bytes. 2) during each cycle there is an equal probability of generating a packet at each of the inputs. 3) packet destinations are uniformly distributed over the outputs. The size of the buffer at each input is 96 bytes. We assume that a packet requests the use of the crossbar two cycles after arriving at the switch. After the arbitration, one cycle is spent before the packet starts to leave the switch [10, 5].

The performance measures used are the average latency, the normalized throughput, and the 99th percentile latency. The latency of a packet is the number of cycles that elapse from when the first byte of a packet arrives at a switch to when it leaves the switch. With ideal arbitration, the minimum latency through a switch is four cycles. The normalized throughput is the average number of bytes received by each output per clock cycle. The 99th percentile latency is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies). It is reported here as a measure of the *fairness* of the arbitration. Lack of fairness in the arbitration results in increasing the 99th percentile latency and increasing the difference between the 99th percentile and average latencies [11]. All the simulations were run for 48,000 cycles. Statistics were gathered only after 16,000 cycles in order to remove start-up effects.

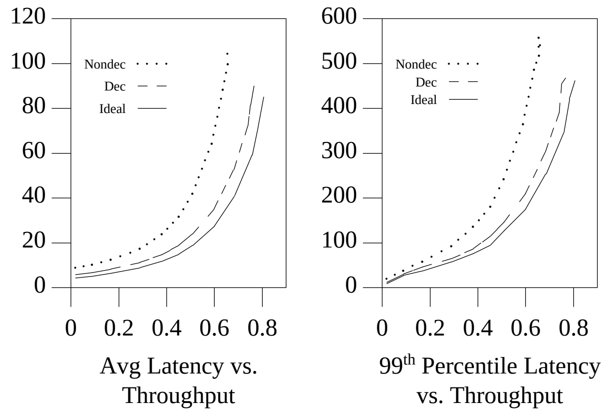

Figure 7: Average latency and 99th percentile latency vs. normalized throughput of a 16×16 switch for three arbitration schemes.

Figure 7 and Figure 8 show the simulation results for a 16×16 crossbar and a 32×32 crossbar. For both switches the decomposed scheme outperforms the nondecomposed scheme by achieving higher maximum throughput, as well as lower average latency and 99th percentile latency. The difference between the decomposed and nondecomposed arbiters increases with increasing crossbar sizes. To understand the reason for this, we consider a crossbar where the nondecomposed arbitration takes m cycles. With the nondecomposed scheme, under very light load, an arriving packet first has to wait, on the average, $m/2$ cycles in order to enter into arbitration and then has to wait for another m cycles for the arbitration to complete. Under heavy traffic load, once a packet leaves an input port buffer, the port has to wait, on the average, $m/2$ cycles before entering arbitration and then wait for another m cycles for arbitration to complete. On the other hand, with the decomposed scheme, under light load the average wait is $m/2$ cycles to enter arbitration and then one cycle for arbitration to

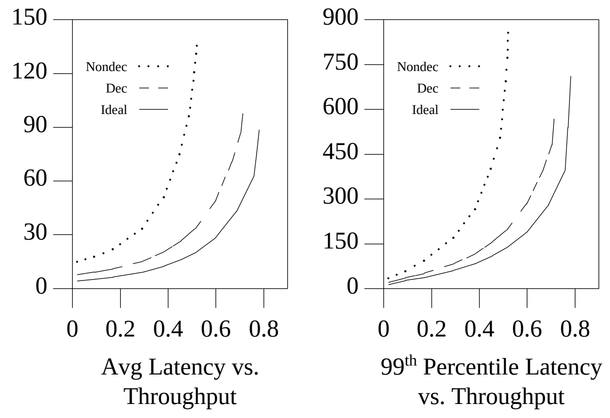

Figure 8: Average latency and 99th percentile latency vs. normalized throughput of a 32×32 switch for three arbitration schemes.

complete. Under heavy load, once a packet leaves an input port buffer, some of the queues at the buffer may enter arbitration immediately in the following cycle and may begin transmission after the one cycle arbitration of their subarray is complete.

It should be noted that for both the decomposed and nondecomposed schemes, performance under heavy load may be improved by using the length of the packet currently being forwarded from an input port to predict when the input port will be free and begin arbitration while the current transmission is still in progress. This technique will involve significant increase in hardware complexity and will not improve performance under light loads. As mentioned earlier, another approach to improving performance is to use tree structured arbiters whose delay increases logarithmically with the size of the crossbar [7]. Based on our simulation results, even the theoretical performance advantage of such arbiters over the decomposed scheme is quite small.

B. Design of Decomposed Arbiters

The decomposed arbiter is composed of several subarrays. Each subarray is implemented as a WWFA. Figure 9 shows the organization of a 2×2 wrapped wave front arbiter [11]. For each arbitration cell, the P (priority) input indicates whether the cell has the top priority. The XI, XO, YI, and YO signals correspond to the W, E, N, and S signals, respectively, of Figure 3. The circular shift register (token ring) is used to rotate top priority among the wrapped diagonals of arbitration cells.

The detailed design of an arbitration cell is shown in Figure 10 and Figure 11. The $\overline{OPB}$ (Output Port Blocked) line is 0 when the output port is blocked. SAE (Subarray Enable) is 1 when the associated subarray is enabled by a signal from the global circular shift register for subarray arbitration (Figure 6). R is the request line from a queue of the buffer and G is the grant line. P (Priority) indicates if the cell gets the top priority. BT (Buffer Transmitting) signal is asserted by the buffer in order to reserve the crosspoint it is using while the packet is being transmitted. When BT is 1, only one R line from the buffer is asserted. $\overline{CB}$ (Column Busy) is a wired OR line

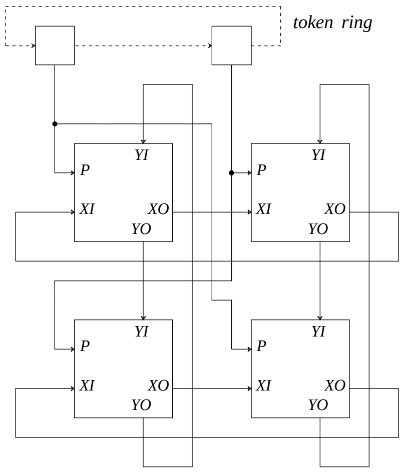

Figure 9: Organization of a 2×2 wrapped wave front arbiter. This can be a subarray of a decomposed arbiter.

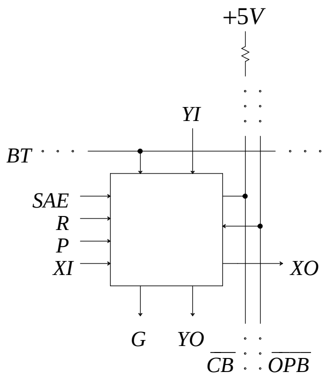

Figure 10: An arbitration cell.

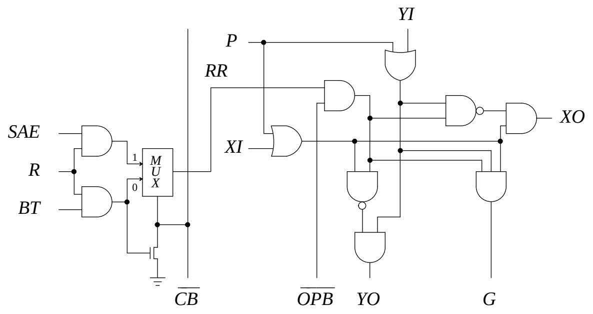

Figure 11: Logic design of the arbitration cell for subarrays of decomposed arbiters.

which connects all the cells in a column, indicating whether some buffer is using the column for an ongoing transmission and it should not be interrupted by other buffer requests.

The circuitry in the right part of Figure 11 is identical to the circuitry of the arbitration cell in a nondecomposed arbiter [11] and implements the following Boolean equations:

$$G = (RR \wedge \overline{OPB}) \wedge (YI \vee P) \wedge (XI \vee P)$$
$$YO = (YI \vee P) \wedge \overline{G}$$
$$XO = (XI \vee P) \wedge \overline{G}$$

The left part of the circuit is unique to the decomposed arbiter. The RR line is asserted when either the particular queue is currently forwarding a packet (R and BT are asserted and $\overline{CB}$ is pulled down), or when the column is idle, the subarray is enabled, and the queue has a packet ready for transmission to the column.

The design presented in this section is amenable for modular implementation where each arbiter subarray is implemented on a single chip and multiple copies of this chip can be used to construct a large arbiter. A single chip with an arbiter subarray as well as a subarray of the data part of the crossbar can be used as a flexible building block for crossbar switches. For example, a 4×4 crossbar switch with byte-wide ports and buffers can use the following pin allocation: 32 (8×4) for input data, 32 (8×4) for output data, 16 for request lines, 4 for BT lines, 16 for grant lines, 16 for XI, XO, YI and YO on the periphery of the subarray, 4 for $\overline{OPB}$, 4 for $\overline{CB}$, 1 for SAE, 2 for the input and output of the local shift register, and some power and clock lines.

To construct a nondecomposed crossbar switch whose size is multiple of the building block chip size, several chips can be connected in an array. The SAE of each chip is set to 1. A decomposed arbiter can be constructed similarly but with the SAE input of each chip is connected to a state latch of the global circular shift register. To provide a decomposed crossbar with the global circular shift register, a latch is included in each chip. Several latches in different chips are linked together to form the global circular shift register. The state of a latch is fed to the associated group of subarrays to indicate ‘‘the turn’’ of the group. This latch adds two more pins to the chip: latch input and latch output (the latch output is also used as the latch state). Since the local circular shift register shifts every n/m cycles for an $n \times n$ crossbar consisting of $m \times m$ subarrays, the clock signal for the local circular shift register can be obtained from the latch output of one latch of the global circular shift register.

IV. Reducing Number of Queues

For a large crossbar, the cost of managing, at each input port, a separate queue for each output port may be prohibitive. The control registers can require significant chip area and the large number of these register will reduce circuit performance (larger decoders, longer buses, additional loads on buses, etc) [5]. This problem can be alleviated by reducing the number of queues in each buffer. The output ports are partitioned into several groups, and all the packets destined to output ports in the same group share the same queue. For example, consider an $n \times n$ switch, with l queues in each input buffer. The output ports are partitioned into l groups. A packet destined to output port k is stored in queue $\lfloor (k\,l)/n \rfloor$. The crossbar arbiter design described in Section III can still be used here even though fewer queues are implemented in each buffer.

Reducing the number of queues in a buffer allows a reduction in the number of wires for *request* and *grant* signals between the buffers and the arbiter. For example, if there are eight queues in a buffer for a 32×32 crossbar, then we need only three wires for the requests from each queue. One wire is used to indicate if there is a request from this queue, and the other two specify which crosspoint is requested. Only one grant wire is needed per a queue. Hence, the total number of wires for *request* and *grant* signals between each buffer and the arbiter is 32. This is a significant reduction compared to the 64 wires that would be required if 32 queues were maintained in each buffer. With this scheme there is additional hardware required in the arbiter for decoding the lines carrying the number of the output port requested by each queue.

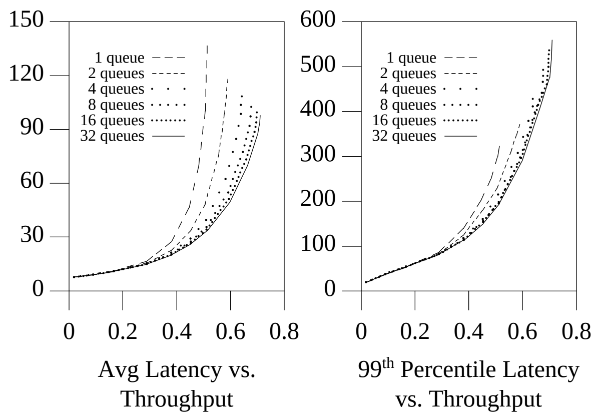

Figure 12: The impact of varying the number of queues per input. A 32×32 switch with 96 byte input buffers.

In order to determine the impact of varying the number of queues per input port, we have simulated a 32×32 switch with 96 byte input buffers. The arbitration is performed by a decomposed arbiter with 4×4 subarrays. The simulation parameters and performance measures are as described in Subsection III.A. Figure 12 shows the results of these simulations. When there are only 16 or even 8 queues, the performance is almost identical to the performance with 32 queues. Since even the performance with only 4 queues is close to the performance with 32 queues, if chip area for buffer management is limited, 4 queues may be a good design choice. Note that for the one queue case, the buffer degenerates into a FIFO buffer.

V. Summary and Conclusions

The time required to arbitrate conflicting requests to crossbar switches increases with the size of the crossbar. We have shown that by decomposing the arbitration process, the crossbar bandwidth can be increased while the average latency is reduced. The additional implementation cost compared to a nondecomposed scheme is minimal. This design lends itself to efficient modular implementation of large crossbars using small crossbar building blocks.

As originally designed, switches based on DAMQ buffers manage, at each input port, a separate queue for each output port. The circuit complexity for managing multiple queues in an input buffer increases as the number of queues is increased. Furthermore, with the original DAMQ design, the number of *request* and *grant* lines between each buffer and the arbiter increases linearly with the size of the switch. Based on these considerations, for a large crossbar, it is undesirable to have at each input port a separate queue for each output port. We have shown that this problem can be alleviated by reducing the number of queues per input buffer. For large switches with limited input buffer size, even if the number of queues is reduced significantly, there is only a small reduction in performance compared to switches that use full DAMQ buffers.

References

1. L. N. Bhuyan, ‘‘Analysis of Interconnection Networks with Different Arbiter Designs,’’ *Journal of Parallel and Distributed Computing* **4**(4), pp. 384-403 (August 1987).
2. W. J. Dally and C. L. Seitz, ‘‘The Torus Routing Chip,’’ *Distributed Computing* **1**(4), pp. 187-196 (October 1986).
3. W. J. Dally, ‘‘Virtual-Channel Flow Control,’’ *17th Annual International Symposium on Computer Architecture*, Seattle, WA, pp. 60-68 (May 1990).
4. D. M. Dias and J. R. Jump, ‘‘Packet Switching Interconnection Networks for Modular Systems,’’ *Computer* **14**(12), pp. 43-53 (December 1981).
5. G. L. Frazier and Y. Tamir, ‘‘The Design and Implementation of a Multi-Queue Buffer for VLSI Communication Switches,’’ *International Conference on Computer Design*, Cambridge, MA, pp. 466-471 (October 1989).
6. M. Kumar and J. R. Jump, ‘‘Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links,’’ *Journal of Parallel and Distributed Computing* **1**(1), pp. 81-103 (1984).
7. T. Lang and M. Valero, ‘‘M-users B-servers Arbiter for Multiple-Busses Multiprocessors,’’ *Microprocessing and Microprogramming*, pp. 11-18 (October 1982).
8. T. Lang, M. Valero, and I. Alegre, ‘‘Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors,’’ *IEEE Transactions on Computers* **C-31**(12), pp. 1227-1234 (December 1982).
9. R. J. McMillen and H. J. Siegel, ‘‘The Hybrid Cube Network,’’ *Distributed Data Acquisition, Computing, and Control Symposium*, pp. 11-22 (December 1980).
10. Y. Tamir and G. L. Frazier, ‘‘High-Performance Multi-Queue Buffers for VLSI Communication Switches,’’ *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 343-354 (May 1988).
11. Y. Tamir and H.-C. Chi, ‘‘Symmetric Crossbar Arbiters for VLSI Communication Switches,’’ Computer Science Department Technical Report CSD-900028, University of California, Los Angeles, CA (October 1990).