On Scheduling Using Parallel Input-Output Queued Crossbar Switches With No Speedup

Saadeddine S. Mneimneh[1], Vishal Sharma[2], Kai-Yeung Siu[1]

Massachusetts Institute of Technology [1]
77 Massachusetts Avenue, Cambrdige, MA 02139
and
Jasmine Networks, Inc.[2]
3061 B, Zanker Road, San Jose, CA 95134

Abstract— **We propose an efficient parallel switching architecture (PSA) that requires no speedup and guarantees bounded delay. Our architecture consists of k crossbar switches operating in parallel under the control of a *single* scheduler, with k being independent of N the number of inputs and outputs of the PSA. Arriving traffic is demultiplexed (spread) over the k identical crossbar switches, switched to the correct output, and multiplexed (combined) before departing from the parallel switch.**

We show that by using an appropriate demultiplexing strategy at the inputs and by applying the *same matching* at each of the k parallel crossbar switches during each slot, our scheme guarantees that the cells of a flow can be read in FIFO order from the output queues of the crossbar switches, thus eliminating the need for cell resequencing. Further, by allowing the PSA scheduler to examine the state of only the first of the k parallel switches, our scheme also reduces considerably the amount of state information required at the scheduler. The scheduling algorithms that we develop are based on existing practical scheduling algorithms for crossbar switches, and have an additional communication complexity that is optimal up to a constant factor.

Our approach also provides a way to build a high capacity switch/router that can support line rates that are *higher* than the speed at which the parallel switches themselves operate.

Keywords— **Scheduling, parallel switches, speedup.**

I. Introduction

TRADITIONAL output-queued or shared-memory architectures are becoming increasingly inadequate to meet the high bandwidth requirements of next generation switches and routers, because having to account for multiple arrivals to the same output requires their switch memories to operate at N times the line rate, where N is the number of inputs.

Although input-queued switches provide an attractive alternative since their memory and switch fabrics may operate at only the line rate, they present a challenge for providing quality-of-service (QoS) guarantees comparable to those provided by output-queued switches, and require a sophisticated scheduler or arbiter, making it a critical component of the switch.

For instance, traditional scheduling algorithms that achieve 100% throughput in an input-queued switch do not provide delay guarantees, and are based on computing a maximum weighted matching [1], [2], [3] that requires a running time of $O(N^{2.5})$ or $O(N^3)$, making them impractical to implement on high-speed switches. Some recent work [4] has, therefore, focused on asking whether an input-queued switch can be made to emulate an output-queued switch, and has demonstrated that this can be achieved by a combination of a speedup in the fabric (of $2-1/N$) and a very clever scheduling algorithm. Such emulation involves substantial book-keeping and communication overhead at the scheduler, however, and, despite its theoretical significance, is not yet practical. Most practical scheduling algorithms for input-queued switches (see, for instance, [7], [8]), require a speedup of between 2 and 4 to achieve adequate QoS guarantees. This means that the switch fabric and the memory need to operate faster than the line rate by the speedup factor. Also, an input-queued switch with speedup requires buffers at the outputs, since more than one cell can now be switched to an output in a given cell slot. Fig. 1 depicts the architecture of a combined input-output queued crossbar switch, where all queues are FIFO queues.

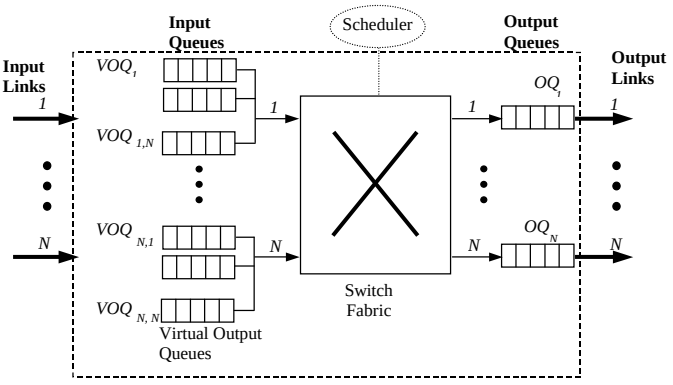

Fig. 1. A combined input-output queued crossbar switch

We propose to use multiple crossbar switches in parallel,

This work was done as part of the first author's summer internship at the Tellabs Research Center, June-August 2000. The work is patented by Tellabs. Vishal Sharma was with Tellabs Research Center in Cambridge, MA. He is now with Jasmine Networks, Inc. The research was partially supported by the Networking Research Program of the National Science Foundation, Award Number 9973015.

allowing each switch to operate at the line rate, so that no speedup is necessary. We show that such an architecture, combined with efficient scheduling algorithms, is both feasible and practical, and that operating the switches in parallel incurs only a small additional computational and communication cost. Furthermore, not only can such an architecture provide bounded cell delay, it can also be used to construct a switch whose aggregate line rate is some multiple of the rate at which any component crossbar switch operates.

II. Motivation

As mentioned in the Introduction, most practical scheduling algorithms in the literature today require a speedup of at least 2. This poses two non-trivial difficulties in moving towards higher speed switches:

- The first is that the memory within the switch must run at a rate faster than that of the external lines. This reduces memory access times, and makes it difficult to build practically useable memories, especially with the continuously increasing line rates.
- The second is that, with speedup, the time available to obtain a schedule by executing the scheduling algorithm is also reduced. Specifically, with a speedup of S, a scheduling algorithm has only $1/S$ units of time in which to compute a schedule, since S scheduling computations must be made per cell slot.

Our approach, therefore, is to eliminate the need for speedup by using crossbar switches in parallel. Our goal is not to emulate output queueing, as was done in some recent works [5]. Rather, it is to obtain an efficient and practical way of achieving basic guarantees, such as bounded cell delay. We also show how our approach, combined with some recently proposed scheduling algorithms, such as the rate matrix based Birkhoff-von Neumann decomposition [6] can be used to support a rate that is k times the line rate of a single crossbar switch in the PSA.

III. Problem Statement

For the purpose of our exposition, we use the parallel switching architecture (PSA) depicted in Fig. 2. The architecture consists of N input ports, each having a demultiplexer, and N output ports, each having a multiplexer. Internally, the PSA consists of k crossbar switches in parallel, with each switch S_i being a combined input-output queued crossbar, like the one depicted in Fig. 1.

At each input port i of the PSA, a demultiplexer forwards cells arriving on that input to one of the k parallel switches. Likewise, at every output port j of the PSA, a multiplexer accesses output queue j for that port in each of the k crossbar switches. During each cell slot, multiple cells may arrive at an input i provided each was destined to a different output j. Thus, the arrival of cells from the same input to different outputs is not restricted. The actual arrival pattern, of course, depends on the traffic shaping used and on the specific implementation of the demultiplexers.

Note that no component in Fig. 2 need run at a rate higher than the line rate. In other words, a demultiplexer forwards at most a cell to a particular VOQ every cell slot, a switch computes one schedule every cell slot, and a multiplexer reads at most one output queue every cell slot.

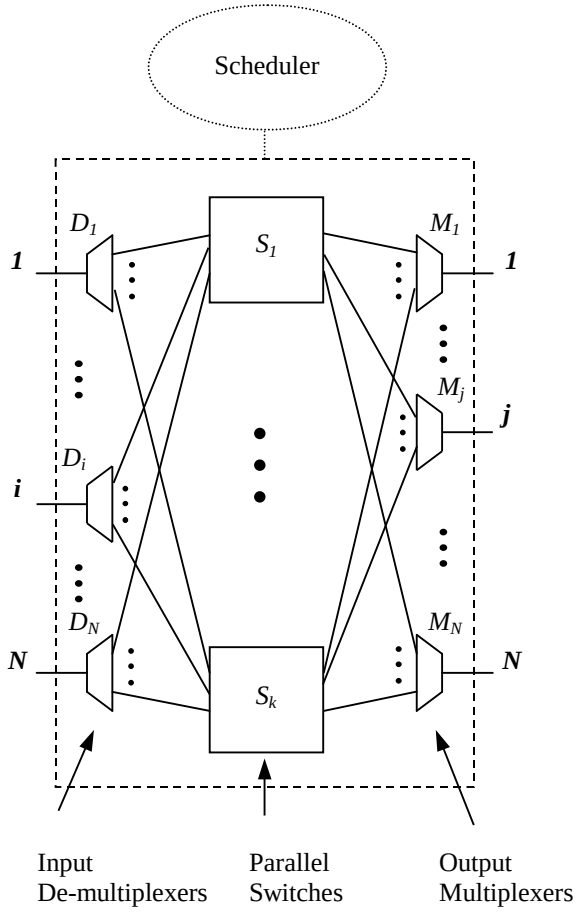

Fig. 2. The parallel switching architecture (PSA).

To proceed further, we define the following notation:

(i, j)	the *flow* (of cells) from input i to output j
$C(i, j)$	a cell from input i to output j
$VOQ^l_{i,j}$	virtual output queue j at input i in switch l
OQ^l_j	output queue j in switch l

Our problem then is to find a scheduling algorithm that provides cell delay guarantees while being efficient and practical to implement. The architecture in Fig. 2 suggests the following natural decompostion of the scheduling algorithm:

- Demultiplexing: At every input i, deciding where to forward each incoming cell.
- Scheduling: In each of the k parallel switches, at each slot, deciding which cells to transmit across the switch.
- Multiplexing: At every output j, deciding which switch S_i, $i = 1, \ldots, k$ to read a cell from.

Before discussing the operation of this architecture, we describe a major issue that could arise when using the PSA, namely overloading the output queues.

Two cells of the same flow that arrive at a given input, and are forwarded to two different crossbar switches may experience different delays depending on the state of each switch, and thus may arrive at the output in the wrong order. Even though it appears that this could be circumvented by controlling the order in which the output queues are read (that is, by determining, at each cell slot, the output queue containing the oldest cell of a flow and reading that cell), there could still be situations, such as the one depicted in Fig. 3, where no output queue at all can be read without violating the order of the cells.

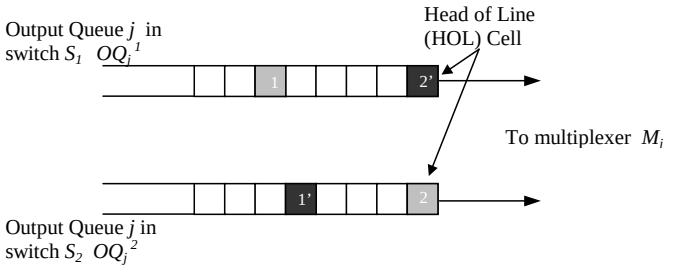

Fig. 3. Possibility of deadlock at the output.

In Fig. 3, the cells at the head of output queue j in both parallel switches are the second cells of their respective flows.

Thus, with FIFO output queues, it is not possible to transfer any cell to output j without violating the order of cells in a flow. Another solution could be to read the Head Of Line cells and temporarily store them for transfer later, when the multiplexer has read deep into the output queues to be able to reconstruct the correct order of cells in a flow. Clearly, if this happens often enough, cell slots will be wasted without delivering cells to output j, causing the FIFO output queues to be overloaded and grow indefinitely. It can be shown (see [10]) that if the parallel switches are allowed to operate independently and the multiplexer waits to reorder cells, the number of cells transmitted at the output could fall consistently short of the number of arriving cells, thereby overloading the output and preventing 100% throughput.

Therefore, on one hand our goal is to enable the switches to operate in a coordinated fashion, while on the other it is to avoid excessive bookkeeping of the type needed in [5] to emulate output queueing.

The key is to avoid the type of deadlock depicted in Fig. 3. To do so, we consider the following two properties:

Definition 1 (Output Contention property) In a single switch, two cells coming from different inputs and destined to the same output cannot be switched during the same cell slot (this is trivial when the switch has no speedup).

Definition 2 (Per-Flow Order property) For any two cells C_1 and C_2 of the same flow, if C_1 arrived before C_2, then by the end of the cell slot during which C_2 was switched, C_1 would also have been switched.

We will show that the above properties are sufficient to ensure that at any output j the cells of a flow can be read in FIFO order. We begin be defining a partial order relation $\prec_{FIFO}$ on cells in the output queues.

Definition 3 (FIFO) For any two cells $C(i,j)$ and $P(k,j)$ at the output, $C(i,j) \prec_{FIFO} P(k,j)$ iff:
- The cell slot during which $C(i,j)$ was switched precedes the cell slot during which $P(k,j)$ was switched, or
- The cell slot during which $C(i,j)$ arrived precedes the cell slot during which $P(k,j)$ arrived, both were switched during the same cell slot, and $i = k$

We call a cell P *early* if there is no cell C such that $C \prec_{FIFO} P$.

Lemma 1: If the **output contention** and **per-flow order** properties are both satisfied, the following is true for every output j of the PSA: At the end of a cell slot, either OQ_j^l is empty for all l or there exists a flow such that its oldest cell over all k switches is an *early* cell and is at the head of OQ_j^l for some l.

Proof: If OQ_j^l is empty for all l, the lemma is true. Assume that there is an l such that OQ_j^l contains an early cell $C(i,j)$. We will prove that $C(i,j)$ is at the head of OQ_j^l and that $C(i,j)$ is the oldest cell of flow (i,j). We first prove that $C(i,j)$ is at the head of OQ_j^l. We know that no cell P can be ahead of $C(i,j)$ in OQ_j^l; otherwise, $C(i,j)$ would not be an early cell because by the *Output Contention* property, P would have been transmitted in a cell slot prior to the cell slot in which $C(i,j)$ was transmitted. Next we prove that $C(i,j)$ is the oldest cell of flow (i,j). If this is not so, then by the *per-flow order* property, we know that the oldest cell of flow (i,j) was transmitted by the end of the cell slot in which $C(i,j)$ was transmitted. This implies that $C(i,j)$ cannot be an early cell, which is a contradiction. ■

The above lemma implies that for every output j, whenever there are cells in the output queues of the parallel crossbars, a cell can be read without violating the FIFO order of cells of a flow (i,j). This eliminates the deadlock situation described earlier and prevents the output queues from being overloaded.

Since the *output contention* property is trivially satisfied when the switches have no speedup, we only need to design our scheduling scheme to satisfy the *per-flow order* property. To do so, we restrict the scheduler to operate as follows. During each cell slot, the scheduler computes a matching M and applies it in all the k parallel crossbar switches. Moreover, for every flow, we follow a special policy to forward the cells to the different switches.

IV. The Approach

To specify our approach, we will describe how the PSA carries out the three steps outlined in Section III. We start with a definition.

Definition 4 (*k-parallel* scheduling) In the PSA setting, *k-parallel* scheduling is one where, during each cell slot, the scheduler computes only one matching, M, and applies it in all k parallel crossbar switches.

A. Demultiplexer Operation

To distribute the incoming cells among the k parallel switches, the demultiplexer follows as a special demultiplexing strategy, which we call *minimum length demultiplexing*, as defined below:

Definition 5 (*Min. length* demultiplexing) For every output j, demultiplexer D_i forwards a cell destined for output j to a switch l that had a minimum number of cells in $VOQ^l_{i,j}$ at the end of the cell slot preceding the current cell slot.

We now prove that this strategy together with *k-parallel* scheduling ensures that the k oldest cells for each flow (i,j) are always in distinct crossbar switches.

Lemma 2: If **minimum length** demultiplexing and *k-parallel* scheduling are used, then at the end of a cell slot, the lengths of $VOQ^l_{i,j}$ and $VOQ^s_{i,j}$ differ by at most 1 for any two switches l and s.

Proof: The proof is by induction, is given in [10]. ■

Using Lemma 2, we can now prove the following lemma:

Lemma 3: If **minimum length** demultiplexing and *k-parallel* scheduling are used, then for any flow, at the end of a cell slot, either all cells at the input side are in distinct switches or the k oldest cells at the input side are in distinct switches.

Proof: We give a sketch of the proof, the details appear in [10]. If at the end of a cell slot T, there is some $VOQ^s_{i,j}$ that is empty, then by Lemma 2, $VOQ^l_{i,j}$ has length at most 1 for all l and hence all cells at the input side are in distinct switches. If at the end of a cell slot T, no VOQs are empty, then for the k oldest cells at the input side not to be in distinct switches, it must be that some VOQ, say $VOQ^l_{i,j}$, contains two of the k oldest cells C_1 and C_2, and another VOQ, say $VOQ^s_{i,j}$ contains a cell C_3 that is not among the k oldest cells. The proof consists of showing that this implies that at the end of some cell slot T_0 prior to T, the lengths of $VOQ^l_{i,j}$ and $VOQ^s_{i,j}$ differ by at least two, which contradicts Lemma 2. ■

Theorem 1: If **minimum length** demultiplexing and *k-parallel* scheduling are used, then the **per-flow order** property is satisfied.

Proof: Assume that the property is violated at the end of cell slot T. This means that for some flow (i,j), a cell C_2, that arrived after another cell C_1, was switched during cell slot T while C_1 remained on the input side at the end of cell slot T. C_2 could not have been in the same VOQ, say $VOQ^l_{i,j}$, as C_1 because of the FIFO property of the input and output queues in a single switch. Moreover, since at most one cell $C(i,j)$ can arrive during cell slot T, C_1 was in $VOQ^l_{i,j}$ at the end of cell slot $T-1$, otherwise C_2 could not have been present in the switch during cell slot T. This means that at the beginning of cell slot T, $VOQ^l_{i,j}$ was non-empty. Since we apply the same matching in all switches, it must be that a cell C_0, other then C_1, was at the head of $VOQ^l_{i,j}$. This implies that at the end of cell slot $T-1$, the length of $VOQ^l_{i,j}$ was at least two. By Lemma 2, at the end of cell slot $T-1$, no VOQ was empty and hence C_2 must have been in its VOQ. This means that at the end of cell slot $T-1$, we had at least $k+1$ cells for flow (i,j) at the input side and hence C_2 must be one of the oldest k cells in order not to violate Lemma 3. This puts C_1 and C_0 among the oldest k cells which violates Lemma 3. ■

We can now prove that using *minimum length* demultiplexing and *k-parallel* scheduling cannot overload the output queues.

Corollary 1: If **minimum length** demultiplexing and *k-parallel* scheduling are used, then for every output j, at the end of a cell slot, either OQ^l_j is empty for all l or there exists a flow such that its oldest cell over all k switches is at the head of OQ^l_j for some l.

Proof: Since the *output contention* property is trivially satisfied, the corollary is immediate from Lemma 1 and Theorem 1. ■

We can prove (see [10] for details) that each of the following demultiplexing strategies, when combined with *k-parallel* scheduling, is a *minimum length* demultiplexing.

Round Robin

In this strategy, each demultiplexer keeps N counters, one for each output. Each counter stores the identity of the switch to which a new cell for that output should be forwarded, and all counters start initially at 0. Every time that the demultiplexer forwards a cell for a particular output to the switch specified by the corresponding counter, it increments that counter mod k. This has the nice property of dividing the rate of a flow equally among the k parallel switches. Thus, each switch sustains a rate $r_{i,j}/k$ if the original rate of the flow is $r_{i,j}$.

Round Robin Reset

This strategy is the *Round Robin* strategy, with the following variation. For every flow (i,j), the system keeps track of the number of cells of that flow still residing at the input side of the crossbar switches. Whenever this number becomes zero, the counter at demultiplexer D_i that corresponds to output j is reset to zero. This strategy requires some extra information (to be kept either in the scheduler or in the demultiplexers) in order to correctly reset the counters of the demultiplexers. As described in [10], however, it actually requires the scheduler to keep less in-

formation for coordinating the operation of multiplexers at the output ports, and in some cases, as mentioned later in this paper, it also helps to reduce the amount of state information that the scheduler must look at for computing a matching.

B. Scheduler Operation

In Theorem 1, we have already shown that *minimum length* demultiplexing together with *k-parallel* scheduling satisfies the *per-flow order* property, which (with the *output contention* property) ensures that, for every output j, it is possible to read a cell (if one is available) without violating the order of cells within a flow. In section IV.C, we explain how, during each cell slot, the multiplexer may identify the appropriate queue to read from. Our focus here is to consider how a matching M may be computed to achieve bounded cell delay. We turn our attention first to a class of scheduling algorithms for a single switch that we call *k-serial* scheduling.

Definition 6 (*k-serial* scheduling) In a single switch setting, *k-serial* scheduling is one in which the scheduler applies a matching consecutively k times before computing and applying a new matching.

We can show that any *k-serial* scheduling algorithm with a particular speedup can be *emulated* by a combination of *minimum length* demultiplexing and a $k'-$*parallel* scheduling, where we define *emulation* as follows:

Definition 7 (Emulation) If, using some *k-serial* scheduling algorithm, a cell C is transmitted across the single switch during a cell slot T, then using **minimum length** demultiplexing and a k'-parallel scheduling algorithm in the PSA, the same cell would also have been transmitted across one of the parallel switches by the end of cell slot T.

Theorem 2: If cell arrivals occur only at the beginning of a cell slot, then any *k-serial* scheduling algorithm under a fractional speedup $S = k/c$ can be emulated using **minimum length** demultiplexing and an $\lceil S \rceil$-*parallel* scheduling algorithm.

Proof: The detailed proof appears in [10]. ■

Theorem 3: If a *k-serial* scheduling algorithm under a fractional speedup $S = k/c$ guarantees a cell arbitration delay D_A, then emulating that scheduling algorithm using **minimum length** demultiplexing and an $\lceil S \rceil$-*parallel* scheduling algorithm guarantees a total cell delay of $(\lceil S \rceil + 1)D_A + B$, where B is a constant, if every output reads cells in the FIFO order.

Proof: By Theorem 2, we know that the $\lceil S \rceil$ -*parallel* scheduling algorithm will guarantee a cell arbitration delay D_A. Therefore, at any time, the number of cells destined to an output j that are still at the input side in all switches cannot be more than $\lceil S \rceil D_A$; otherwise, the inputs of some switch l will have more than D_A cells destined to output j, and hence at least one cell will be delayed for more than D_A cell slots. By Corollary 1, if there are cells waiting in some output queue OQ_j^l, then it is possible to read a cell at output j without violating the FIFO order. Consider a cell slot in which some OQ_j^l becomes non-empty for the first time. At the beginning of this cell slot, we know that the number of cells destined to output j that were still at the input side in all the switches was at most $\lceil S \rceil D_A$. If in every cell slot during the next c cell slots, some OQ_j^s is non-empty, then we know that by the end of the c cell slots we will have read c cells at output j. However, during the c cell slots, the total number of cells that could have been transmitted to some output queue of port j is $\lceil S \rceil D_A + c + B$, because at most $c + B$ cells destined to output port j could have arrived during c cell slots, where B is a burst constant. This means that the total number of cells that remain in the output queues of port j after c cell slots is at most $\lceil S \rceil D_A + B$. This is true for any c; therefore, at any time, the number of cells in all output queues of port j cannot be more than $\lceil S \rceil D_A + B$. As a result, since the output emulates a FIFO queue, once a cell arrives at the output side, it will be delivered within at most $\lceil S \rceil D_A + B$ cell slots, hence guaranteeing a total cell delay of $(\lceil S \rceil + 1)D_A + B$. ■

If *round robin* demultiplexing is used, then to guarantee a total cell delay, we need not restrict the output to read cells in the FIFO order. We only require that the output read cells of the *same flow* in order, which is a requirement we have imposed throughout in this paper.

Theorem 4: If a *k-serial* scheduling algorithm under a fractional speedup $S = k/c$ guarantees a cell arbitration delay D_A, then emulating that scheduling algorithm using *Round Robin* demultiplexing and an $\lceil S \rceil$-*parallel* scheduling algorithm guarantees a total cell delay of $(\lceil S \rceil + 1)D_A + B + (\lceil S \rceil - 1)N - 1$, where B is a constant and N is the size of a switch, provided every output reads cells of the same flow in order.

Proof: The proof is similar to the proof of Theorem 3, and appears in [10]. ■

It remains for us to show the existence of *k-serial* scheduling algorithms that guarantee a cell arbitration delay under a speedup $S = k/c$. It turns out that we can do so by modifying some existing scheduling algorithms that guarantee cell arbitration delay under some speedup S and make them *k-serial* scheduling algorithms.

B.1 Some *k-serial* Scheduling Algorithms

To convert an existing scheduling algorithm for a single switch into a *k-serial* scheduling algorithm, we simply modify the existing algorithm to hold a matching that it computes constant for k cell slots. Our motivation is that the state of a switch cannot change substantially within a constant time. Thus, holding the same matching for k slots should still be able to guarantee a cell arbitration delay.

We were able to prove this fact for several existing scheduling algorithms, such as the oldest cell first OCF due to Charny et al [7], the central queue algorithm due to Kam et al [9], and a special maximal matching algorithm. In each of these algorithms, holding the matching for a constant time adds an additional delay of $O(\frac{kN}{1-\alpha})$, where k is the number of times we hold the matching, N is the size of the switch, and α is the traffic rate per input. We refer the interested reader to [10] for details.

B.2 Birkhoff-von Neumann Decomposition: A *k-parallel* Algorithm

Chang et al [6] have proposed an algorithm that is capable of providing service guarantees for input-buffered crossbar switches with no speedup. The algorithm consists of taking a static rate matrix and computing a schedule in time $O(N^{4.5})$, based on a decomposition result of Birkhoff and von Neumann. The schedule is a static list of matchings, corresponding to permutation matrices obtained from a decomposition of the rate matrix, that are applied according to certain weights.

In our context, we may utilize this algorithm in conjunction with round robin demultiplexing, which ensures that the rate matrix of any crossbar switch is equal to the original rate matrix scaled by k. Since all parallel crossbars have the same rate matrix, the same schedule can be applied to them. Thus, as a natural consequence of this approach, we can apply the same matching in all of the parallel crossbars of the PSA. The result is that our system now provides the exact same guarantees as the original algorithms of Chang, with the added advantage, that we can sustain a line rate that is k times the rate at which the parallel crossbars operate.

B.3 Reducing State Information

It can be shown (see [10]) that when the speedup S is an integer, the *k-parallel* scheduler can reconstruct, from the state of the parallel crossbars of the PSA, the state of the single switch running the *k-serial* scheduler. This requires, however, that the PSA scheduler examine the state of each of the k parallel crossbars, and maintain global state. It turns out that this global state requirement can actually be eliminated.

For the single-switch scheduling algorithms discussed above, only two kinds of state are needed: the oldest cell of every flow, and the length of each VOQ. Details on reducing the state information for the different algorithms mentioned above appear in [10]. We will describe here how to reduce the state information for OCF, which requires the oldest cell of each flow. By using round robin reset demultiplexing, we ensure that the oldest cell of every flow is always in the first switch. Thus, when using OCF and round robin reset the scheduler need only look at the state of the first switch to compute a matching.

Observe that state reduction is not an issue for the Birkhoff von Neumann decomposition algorithm, because it only stores a pre-computed schedule and so does not require any state information from the crossbar switches for its operation. Therefore, it does not require the use of round robin reset.

C. Multiplexer Operation

We have already shown that by using minimum length demultiplexing and *k-parallel* scheduling, it is possible for the multiplexer at an output port of the PSA to always read a cell from the output queues of the parallel crossbars such that the order of cells of a flow is not violated. The only question that remains is how a multiplexer M_j determines which output queue OQ_j^l to read the next cell from? This may be done as follows.

For each output j of the PSA, the scheduler maintains a FIFO list $\mathcal{L}_j$ of tuples (c, s) for each cell slot T, where during cell slot T, c is the number of cells switched from input i to output j, and s is the index of the parallel crossbar whose j^{th} output queue contains the oldest cell among the switched cells of flow (i, j). During each cell slot for which flow (i, j) belongs to a matching M, the scheduler adds (c, s) to $\mathcal{L}_j$. The scheduler may easily obtain the information to do so from the demultiplexers.

As shown in [10], the total communication required between the demultiplexers and the scheduler is $N\lceil \log N \rceil + 2N\lceil \log k \rceil$ bits. At the output side, each multiplexer M_j periodically retrieves from the scheduler a tuple (c, s) from which it learns the number of cells that must be read and the identity s of the crossbar switch from whose output queue the multiplexer must start reading the first cell of this round. The communication between the scheduler and output multiplexers requires $2N\lceil \log k \rceil$ bits. Hence, the total communication with the scheduler is $O(N \log kN)$, which is within a constant factor of the $\Omega(N \log N)$ bits needed to specify a matching in a single switch.

V. Conclusion

We suggested a scheme that eliminates the need for speedup by using $\lceil S \rceil$ parallel crossbar switches, where S is the speedup of the original switch. The key to our approach was to use a common scheduler and apply the same matching in all the parallel switches. We proved that this is equivalent to simply holding the same matching constant for some number of slots in the single switch setting, which still guarantees bounded cell delay for a number of known algorithms. In addition, both demultiplexing and multiplexing at the inputs and outputs of the PSA, respectively, could be done using $O(N \log kN)$ bits of communication between the scheduler and the parallel crossbars compared the $\Omega(N \log N)$ bits of communication needed in a single switch.

The choice of FIFO queues in this work was based on their ease of implementation at high speeds. A direction for future work could be to relax the assumption that the output queues in a parallel crossbar are FIFO queues, thus allowing for out-of-order delivery of cells. The goal then would be to bound the extent to which a cell could be "out-of-order." In other words, to bound the number of undelivered cells that arrived before a delivered cell. Another direction of work would be to consider switches with

shared memory modules (which would be similar architecturally to the model in [5], since an output-queued switch can be viewed as a shared memory module), and try to devise practical and scalable scheduling algorithms that provide throughput and/or delay guarantees.

References

[1] A. Mekkittikul and N. McKeown, *A starvation-free algorithm for achieving 100% trhoughput in an input queued switch*, Proc. Int'l Conf. Computer Commun. Networking, ICCCN'96, pp. 226-231, October 1996.

[2] A. Mekkittikul and N. McKeown, *A practical scheduling algorithm to achieve 100% throughput in input-queued switches*, Proc. IEEE INFOCOM'98, vol. 2, pp. 792-99, March-April 1998.

[3] N. McKeown, V. Anantharam, J. Warland, *Achieving 100% throughput in an input-queued switch*, Proc. IEEE INFOCOM'96, vol. 1, pp. 296-302, March 1996.

[4] S.-T. Chuang, A. Goel, and N. McKeown, *Matching output queuing with combined input output queued switches*, IEEE J. Select. Areas in Commun., vol. 17, no. 6, pp. 1030-39, June 1999.

[5] S. Iyer, A. Awadallah, and N. McKeown, *Analysis of a packet switch with memories running slower then the line rate*, Proc. IEEE INFOCOM'2000, vol. 2, pp. 529-37, March 2000.

[6] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, *On service guarantees for input-buffered crossbar switches: A capacity decomposition approach by Birkhoff and von Neumann*, Proc. Seventh Int'l Workshop on Quality of Service, IWQoS'99, pp. 79-86, May-June 1999.

[7] A. Charny, P. Krishna, N. Patel, et al, *Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup*, Proc. Sixth Int'l Workshop on Quality of Service, IWQoS'98, pp. 235-44, May 1998.

[8] P. Krishna, N. S. Patel, and A. Charny, *On the speedup requirement for work-conserving crossbar switches*, IEEE J. Select. Areas in Commun., vol. 17, no. 6, pp. 1057-69, June 1999.

[9] A. Kam, K.-Y. Siu, and R. Barry, *A cell switching WDM broadcast LAN with bandwitch guarantee and fair access*, IEEE/OSA J. Lightwave Technol. vol. 16, no. 2, pp. 2265-80, Dec. 1998.

[10] S. Mneimneh and V. Sharma, *Scheduling using parallel crossbar switches with no speedup,* Tellabs Research Center Report, TRC-00-06, Tellabs Operations, Inc., September 2000.