A Scalable Memory-Efficient Architecture for Parallel Shared Memory Switches

Brad Matthews, *Student Member, IEEE*, Itamar Elhanany, *Senior Member, IEEE*
Networking Research Group
Electrical and Computer Engineering Dept.
University of Tennessee
Knoxville, TN 37996

***Abstract*— Parallel shared memory (PSM) switch architectures were initially introduced as means of resolving the high memory bandwidth requirements imposed by output-queued switches. At the core of the PSM architecture is a memory management algorithm that determines, for each arriving packet, the memory unit in which it will be placed. Recent work has indicated that in order to achieve high throughput, the number of parallel memories needed is $O\left(N^{1.5}\right)$, thereby significantly limiting scalability. This paper introduces a novel pipelined memory management algorithm which maintains a computational complexity of $O\left(1\right)$ while reducing the number of required parallel memories to $O\left(N\right)$. Our goal is to extend existing shared-memory architecture results in the context of Fabric on a Chip (FoC) - a paradigm that advocates the consolidation of core packet switching functions on a single chip. A detailed discussion is provided pertaining to the fundamental properties of the proposed scheme, along with hardware implementation considerations that illustrate its scalability and performance attributes.**

I. Introduction

Output-Queued (OQ) switch architectures are known to offer distinct performance advantages with respect to input-queued schemes. Minimal average packet delay, controllable Quality of Service (QoS) provisioning and work conservation under a broad range of admissible traffic conditions represent several of the desirable characteristics. However, OQ switches lack scalability due to the high memory bandwidth constraints they impose. These constraints are derived from the requirement that each memory in an $N \times N$ switch, where N is the number of ports, must support up to N write operations and one read access in a single cell time. This results in a memory bandwidth requirement of $O\left(NR\right)$, where R is the line rate, and an internal speedup of N. Clearly, when increasing port densities and line rates, this represents a significant impediment in the design of scalable OQ switching platforms.

The Fabric on a Chip (FoC) [1] approach strives to leverage recent advances in chip density, packaging technology and the availability of large on-chip memory resources to enable the consolidation of switching functions on a single chip. This yields several key advantages. First, the need for virtual output queueing (VOQ) [2] as well as some output buffering associated with standard input-queued switch architectures, is eliminated. Second, the need for external memory devices is reduced by exploiting the ability to access multiple on-chip memories. Moreover, the crosspoint switches and scheduler, pivotal components in input-queued switches, are avoided thereby substantially reducing chip count and power consumption. Third, much of the signaling and control information that typically spans multiple chips can be carried out on a single chip. Finally, the switch management and monitoring functions can be made centralized, since all information is available at a single location.

This paper extends previous work by the authors [1] on the design of large-scale PSM switches, from a single-chip realization perspective. By introducing a column-based packet placement framework, a more efficient high-speed memory management algorithm is obtained, thereby reducing the memory requirement from $O\left(N^{1.5}\right)$ to $O\left(N\right)$.

The rest of the paper is structured as follows. Section II provides an overview of prior work pertaining to scaling PSM switches. In section III, the proposed architecture is described and analyzed. Section IV extends the basic scheme by introducing speedup and load-balancing mechanisms, while in section V the conclusions are drawn.

II. Prior Work on Scaling PSM Switches

The foundations of the FoC approach are tightly coupled with the PSM architecture proposed in [3]. The letter utilizes a pool of slow-running memory units operating in parallel to sidestep the high memory bandwidth requirements of OQ switches, while retaining their desirable performance attributes. Initial work has indicated that, assuming each of the shared memory units can perform at most one packet-read or -write operation during each time slot, the sufficient number of memories needed for a PSM switch to emulate a FCFS OQ switch is $K = 3N - 1$ [3]. The latter can be proven by using constraint sets analysis (also known as the "pigeon hole" principle).

At the core of the PSM architecture is a memory management algorithm that determines, for each arriving packet, the memory unit in which it will be placed. Attempting to simplify this algorithm has been the focus of recent studies. In [4],[5] Prakash, Sharif, and Aziz proposed the Switch-Memory-Switch (SMS) architecture, which is a variation on the PSM switch, as an abstraction of the M-series Internet core routers from Juniper. The approach consists of statistically matching input ports to memories, based on an iterative algorithm that statistically converges in $O(logN)$ time.

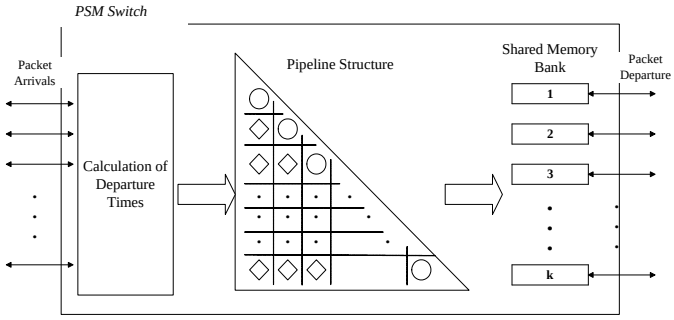

Fig. 1. General architecture of the row-based parallel shared memory (PSM) switch. Arriving packets are placed in a set of $(k > N)$ memory units.

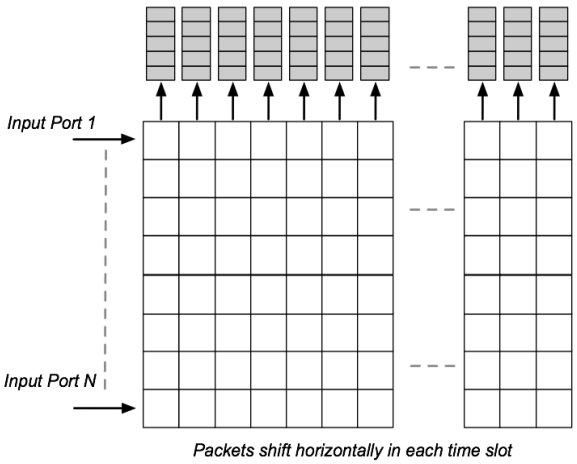

Fig. 2. Packets are selected for placement into each column memory based on their departure time.

However, in this scheme, each iteration comprises multiple operations of selecting a single element from a binary vector. Although the nodes operate concurrently from an implementation perspective, these algorithms are $O(log^2 N)$ at best (assuming $O(logN)$ operations are needed for each binary iteration as stated above). Since timing is a critical issue, the computational complexity should directly reflect the intricacy of the digital circuitry involved, as opposed to the high-level algorithmic perspective. A similar iterative approach has been taken in [6], having the same drawbacks.

In prior work [1], the authors have proposed a row-based, pipelined packet placement architecture for PSM switches, depicted in Figure 1. A key attribute of this approach is the reduction of computational complexity to $O(1)$. However, the subsequent cost associated with reducing the placement complexity is an $O\left(N^{1.5}\right)$ memory requirement and a fixed processing latency. The memory management algorithm utilizes a row-based placement scheme, whereby a pipeline with L rows has a single unique memory associated with each row. The pipeline architecture consists of $\frac{L(L+1)}{2}$ cell buffering units arranged in a triangular structure. Each row is associated with one of the parallel shared memory units, such that L parallel shared memories are required. It was proven that for $k^2 = N$, a sufficient bound on the number of memories is given by $L(k) = 4k^3 - 5k^2 + k + 1$, suggesting an $O\left(N^{1.5}\right)$ memory requirement.

A consequence of the high memory requirement is the significant amount of cell buffering resources required to construct a pipeline with $O\left(N^{1.5}\right)$ rows. This is certainly feasible for low port densities, but limits scalability. Moreover, he complexity of the packet placement process also limits system scalability. This paper aims to overcome the aforementioned issues by taking a column-based approach to the packet placement process.

III. Column-based Packet Placement Algorithm

A. Switch Architecture

This section provides a detailed description of the proposed memory management algorithm for PSM switches. The pipeline architecture, illustrated in Figure 2, consists of $M \times N$ cell buffering units arranged in a rectangular structure. Prior to cell arrival at the pipeline structure, a departure time must be determined. In the context of this paper, we will consider a first-come-first-serve (FCFS) scheduler by which packets are assigned departure times in accordance with their arrival order. To provide delay and rate guarantees, more sophisticated schedulers [7] can be incorporated which is reflected by the departure time assignments. Regardless of the scheduling algorithm selected, the focus of this paper is on the memory management algorithm that distributes the packet-placement process, at a cost of fixed delay.

The pipeline architecture consists of $M \times N$ cell buffering units arranged in a rectangular structure. Each column is associated with a single parallel memory unit. Hence, the architecture requires a total of M parallel memories. Cells arrive at the leftmost column with the packet at input port i initially inserted into row i. The underlying mechanism is that cells are shifted to the right every time slot. A placement unit in each column determines whether a cell located in its column can be placed in the corresponding column memory. A cell will be placed in the associated column memory if the following conditions are met: (1) the memory associated with the column in which it is currently located does not already contain a packet with the same departure time; (2) the packet is selected for placement over the potential $N-1$ other packets with the same arrival time. As a cell progresses through each subsequent stage, memory placement contentions are reduced. In this scheme, once a cell is shifted to the last column of the pipeline, it is guaranteed to be placed in a memory that does not contain a packet with a matching departure time.

B. Basic Analysis of Resource Requirements

In the proposed architecture, rows can be viewed as simple shift registers, whereby cells are shifted one stage to the right at each time step. At each stage of the pipeline, a single cell assignment is attempted per column. The motivation for this approach is to reduce the complexity of the placement

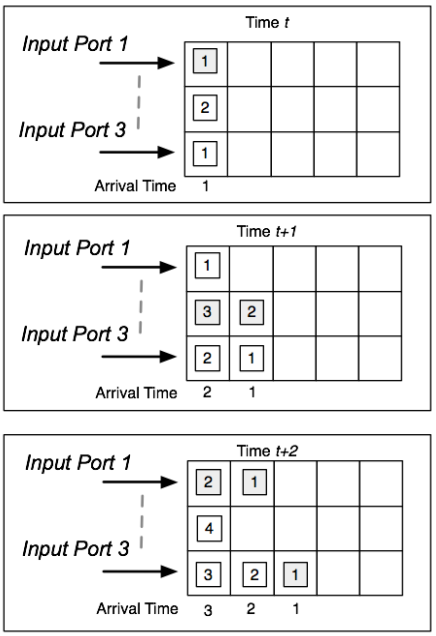

Fig. 3. Example illustrating the proposed memory management algorithm for a 3-port switch. The state of the pipeline structure is depicted for 3 consecutive time slots.

algorithm by isolating memory assignments, thus minimizing memory contention.

Theorem 1: A total of $2N-1$ column memories is sufficient for an N-port switch utilizing the proposed packet placement algorithm.

Proof: Consider an arriving cell, C_1. It will find at most $N-1$ other packets with identical arrival time competing for the same memory. Furthermore, at most $N-1$ other packets with the same departure time could have been placed in unique memories prior to the arrival of C_1. Thus, at least $(N-1)+(N-1)+1 = 2N-1$ memories are required to guarantee the placement of cell C_1. ■

Each column maintains a mapping that defines memory locations, corresponding to departure times, which have been reserved by cells successfully placed in memory. This mapping, which is effectively a binary mask, shall be referred to as the column's *occupation vector*. Each column additionally maintains a mapping that specifies pre-allocated (or reserved) departure times of cells that have yet to be placed in a column memory. This mapping is referred to as the column's *vacancy vector*. The placement element in the pipeline performs a bitwise xor operation over the occupation andvacancy vectors to determine if there is a cell with a departure time available for placement in the memory. If a cell is available and selected, it will be extracted from the pipeline and placed in the corresponding column memory. Cells not selected for placement are subsequently forwarded to the next stage of the pipeline.

Example 1: Consider the simple scenario depicted in 3. The state of the pipeline for three consecutive time slots (i.e. t,$t+1$, $t+2$) is shown. At time t, three cells are inserted into the first column of the pipeline with departure times $\{1,2,1\}$ respectively. Assuming that all memories are initially empty, the placement engine will simply select the first cell in the column for placement. Thus, the cell in row 1 with departure time 1 is placed in the memory associated with column 1. The remaining cells $\{2,1\}$ will shift right to column 2 at time $t+2$, while new cells, with departure times $\{1,3,2\}$, are inserted into the first column. The placement element at column 2 will now select the cell with departure time 2, leaving the only remaining cell, $\{1\}$, to shift to column 3. In column 1, a cell with departure time 1 has already been placed in the memory associated with column 1. Thus, the placement element in column one must select the cell at row 2, with departure time 3, for placement. At time $t+2$, additional cells, with departure times $\{2,4,3\}$ are introduced into the pipeline at column 1. As the cell with departure time 2 has yet to be selected, it is next selected for placement. Additionally, columns 2 and 3 both select cells with departure time of 1.

It can be observed from this example, that the column-based memory management algorithm clearly provides an effective mechanism for resolution of memory contention.

Scalability is achieved as decisions are disassociated from the number of ports in the system. Increased port densities does not directly infer an increase in the decision time. Placement decisions are determined only by the number of departure times offered by the switch fabric in the form of the occupation and vacancy vectors. In the context of FoC, these mapping vectors are bound only by the depth of an individual column memory.

In all practical switching systems, once a buffer approaches (or is close to approaching) its limit, flow-control signaling should be provided to the traffic sources, indicating the need to either slow down or temporarily stop the flow of packets to a particular destination. Such a mechanism is always required since instantaneous data congestion may occur at any router or switch. In fact, even if the traffic is said to be statistically admissible, implying that no input or output is oversubscribed, it may still be the case that for short periods of time a given output port is oversubscribed. To address such scenarios, and in an effort to reduce the probability of packet loss, the linecards typically host large memory spaces.

IV. Enhancing the Architecture

A. Load Balancing Memory Usage

To this point, we have assumed that packets are inserted into the memory management pipeline at the leftmost column and are shifted right at each time slot. The placement elements for each column select a single cell for storage in the associated column memory. As cells progress in the pipeline, their placements tends to be concentrated in the leftmost columns. This is intuitive considering the observation that for every cell in a given pipeline column, there must be a corresponding cell in that column's memory with a matching departure time. After $N-1$ successive occurrences where no packets are

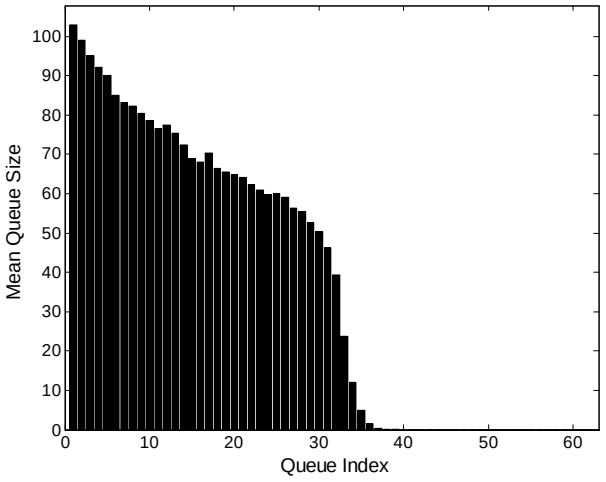

Fig. 4. Queue size distribution for a PSM system without load balancing.

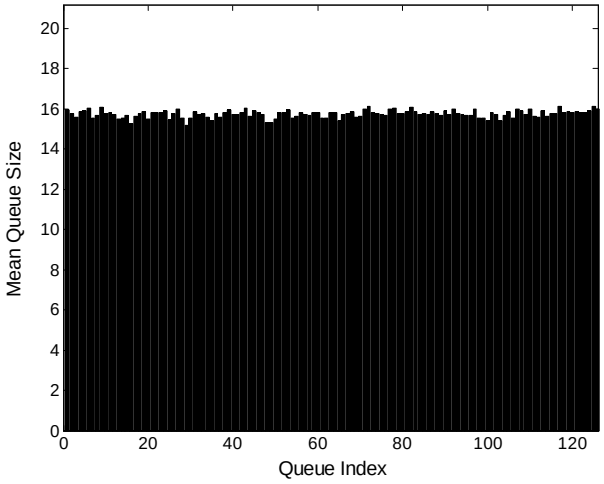

Fig. 5. Queue size distribution with load balancing.

placed, at least one packet is guaranteed to be placed in the successive N columns.

In Figure 4, a 64-port switch with 127 memories is simulated with a normalized traffic load of one. Arrivals are uniformly distributed across the destinations and obey a Bernoulli i.i.d. process. The goal of this exercise was to depict the queue distribution under heavy traffic load conditions. As evident from the queue distribution presented, cell placements are denser in the first N columns.

In order to evenly distribute the cells across the memories, a desirable goal from an implementation perspective, the elements at which cells are inserted into the pipeline must vary for each port. To accomplish this goal, a rotation technique is employed, whereby cells arriving at port i are inserted into the column that precedes the insertion point used in the previous time slot. For example, if at time t a cell was inserted into column l, then all cells inserted at time $t+1$ must be placed in column $l-1$. I If a cell is inserted into the rightmost column and has yet to be placed in memory, it is shifted to the leftmost column, representing a circular shift, to avoid cell loss. As cells advance in a clockwise manner and insertion points occur counter-clockwise, then we must ensure that a cell will not meet an insertion point within $2N-1$ time slots to avoid cell loss.

Theorem 2: Uniform distribution of cells across the memories can be achieved with $4N-2$ memory units.

Proof: For a cell C_1 inserted at column L_1, there must be $2N-1$ columns prior to the insertion of another cell. Thus, an additional $2N-1$ columns are required to rotate the insertion points to ensure uniform distribution across the memories. Aggregating the two terms, we conclude $4N-2$ memories are sufficient to achieve a balanced cell distribution. ■

In Figure 5, a 32-port switch with 126 memories is simulated assuming maximal load. It is observed that a rather uniform memory occupancy distribution results, with fewer cells stored in each memory.

B. Computation and Memory Speedup

To achieve additional reduction in the number of required memories, we must reduce memory contention in the packet placement process. One source of contention is a blocked placement that occurs as a result of offering only one memory location per departure time for each memory. We can alleviate this constraint by provisioning multiple cell placements, allowing a memory to host $1 < m \leq N$ cells with identical departure times. As a result, a given cell, with departure time d, will be considered blocked by a memory if the latter already contains m cells with departure time d.

Theorem 3: It is guaranteed that $2N-m-1$ memories are sufficient for an $N-$port switch, where each memory can hold m cells with the same departure time.

Proof: Consider the arrival of cell C_1 to the first column of the switch. It will find at most $N-1$ other packets with the same arrival time competing for the same memory. Moreover, there are at most $N-m-1$ other packets with the same departure time that may have been placed in a unique memory prior to the arrival of C_1. Therefore, a total $2N-m-1$ memories are required to ensure placement of C_1. ■

It should be noted that the timing for establishing a memory's availability by a placement element is unaffected by the inclusion of multiple cell placements. The primary trade-off resides in the increased density of each memory unit and the requirement that packets must be read from memory at a rate m prior to being forwarded to the appropriate egress ports. Having made this observation, we introduce a placement (or computation) speedup of s, which assumes that the pipeline operates at a rate s times faster than the line rate. One advantage of operating the pipeline at an increase rate is reduced latency. Moreover, cells from a set of N inputs can be presented to the switch fabric in groups consisting of $\frac{N}{s}$ cells. This further provides a reduction in the number of conflicts

TABLE I

NUMBER OF MEMORIES IN THE PROPOSED PSM SWITCH

Switch Ports (N)	Speedup (s)	Placements (m)	Memory Units
16	1	1	31
16	2	2	21
16	4	4	15
32	1	1	63
32	2	2	45
32	4	4	37
64	1	1	127
64	2	2	93
64	4	4	75

associated with packets possessing the same arrival time form N to $\frac{N}{s}$.

Theorem 4: A total of $\left(\frac{s+1}{s}\right) N - 1$ memories is sufficient for an N-port switch with a placement speedup of s.

Proof: For a placement speedup of s, an arriving cell, C_1, will find at most $\frac{N}{s} - 1$ other packets with the same arrival time competing for the same memory. Furthermore, at most $N - 1$ other packets with the same departure time may have been placed in unique memories prior to the arrival of C_1. Therefore, at total of $\left(\frac{s+1}{s}\right) N - 1$ memories are required to ensure placement of cell C_1. ■

Given that columns are interlocked with memories, the incorporation of placement speedup infers a memory write speedup equal to s. Recall that multiple cell placement does not infer an increase in memory write speed and the pipeline can operate at the line rate. Thus, we can utilize both multiple cell placements and placement speedup concurrently to gain an even greater reduction in memory requirements. In view of the above assertions, it should be evident that the sufficient number of memories in this case would be $\left(\frac{s+1}{s}\right) N - m - 1$. Table I outlines the number of memories needed for various port densities, and m and s values.

C. Hardware Implementation Considerations

It becomes evident that the critical path in the design is the placement decision that must be made by the placement elements in each column. Each must determine whether a cell, residing in column c_i $(i \in [1, 2, ...M])$, can be placed in the associated column memory. At each column in the pipeline, cells that have yet to be placed are inspected, by their departure time, to locate potential candidates for placement. If a cell is determined viable, (i.e. the corresponding column memory does not contain a cell with a the same departure time), and is selected for placement, it will be extracted from the pipeline and written to the memory. Otherwise, the cell will advance to the next column.

To determine the feasibility of placing a cell in a column memory, each memory maintains a binary occupation vector,of length k, denoting the depth of the memory, to indicate departure times that are currently reserved. To determine which cells in a given column are available for placement, a requests vector is created as cells are inserted into the pipeline. This vector, also of length k, provides a bitmap corresponding to the departure time for each of the cells located in a column. By performing a bitwise xor operation of the two vectors, the set of viable cells (referred to as the candidates vector) is obtained. A priority encoder can then be used to select a cell for placement into the corresponding column memory. The result of this decision process is that c_i is either written to the memory associated with the column in which it resides or shifted to the next stage of the pipeline.

Given that the xor operation can be achieved at high speed, it becomes apparent that the priority encoder is the predominately time-consuming function. The complexity of a priority encoder is generally acknowledged to be $O(logN)$, where N denotes the number of elements at its input. It is noted that this complexity pertains to 2 -bit-level operations (rather than more complex arithmetic abstraction), clearly suggesting that the method is very efficient from a computational standpoint.

V. CONCLUSIONS

This paper introduced and analyzed an architecture for designing scalable parallel shared memory switches. In the context of emulating an output-queued switch, it has been argued that a fundamental challenge pertains to the memory-management algorithm employed by PSM switches. A column-based packet placement algorithm and related high-speed parallel architecture were described in detail, emphasizing the feasibility attributes of the proposed approach. The switch model and framework presented here can be broadened to further investigate the concept of consolidating multiple switch fabric functions on silicon.

VI. ACKNOWLEDGEMENTS

This work has been partially supported by the Department of Energy (DOE) under research grant DE-FG02-04ER25607, and by Altera, Inc.

REFERENCES

[1] B. Matthews, I. Elhanany, and V. Tabatabaee, "Fabric on a chip: Towards consolidating packet switching functions on silicon," in *Proc. IEEE International Conference on Communications (ICC)*, June 2006.

[2] Y. Tamir and G. Frazier, "Higher performance multiqueue buffers for vlsi communication switches," in *15th Annual Symposium on Computer Architecture*, pp. 343–354, 1988.

[3] S. Iyer, R. Zhang, and N. McKeown, "Routers with a single stage of buffering," 2002.

[4] A. Aziz, A. Prakash, and V. Ramachandra, "A near optimal scheduler for switch-memory-switch routers," in *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 343–352, July 2003.

[5] A. Prakash, A. Aziz, and V. Ramachandra, "Randomized parallel schedulers for switch-memory-switch routers: Analysis and numerical studies," IEEE INFOCOM 2004, 2004.

[6] Y. Xu, B. Wu, W. Li, and B. Liu, "A scalable scheduling algorithm to avoid conflicts in switch-memory-switch routers," in *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, pp. 57–64, Oct. 2005.

[7] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *Proc. of ACM SIGCOMM '95*, pp. 231–242, September 1995.