Techniques for Fast Shared Memory Switches

Sundar Iyer and Nick McKeown
Computer Systems Laboratory, Stanford University
Stanford, CA 94305-9030
{sundaes, nickm}@stanford.edu

Abstract -- **Shared memory is commonly used to build output queued (OQ) switches. An OQ switch is known to maximize throughput, minimize delay and can offer QoS guarantees. However it is generally believed that high capacity switches cannot be built from shared memory switches because the requirements on the memory size, memory bandwidth and memory access time increase linearly with the line rate R and the number of ports N. In this paper, we ask the following question: Is it possible to build a high-speed shared memory switch in which the individual memories (in this case, DRAMs) operate at slower than the line rate? We show that this is indeed possible i.e. a shared memory switch with a specific "conflict free DRAM memory management algorithm" (CFDMMA) can emulate a first-come-first-served OQ switch if each memory operates at a rate of approximately $3NR/k$, where k is the number of DRAMs used in parallel.**

We also show that the switch can support qualities of service, if the memories operate at a rate of $4NR/k$. To enable this, we introduce a small fixed sized co-ordination buffer of size Nk and prove that the shared memory switch can emulate a OQ switch supporting QoS within a relative delay bound of Nk time slots.

Keywords--**packet-switch; shared memory, packet buffers, output-queueing;**

I. INTRODUCTION

Several different architectures are commonly used to build packet switches (e.g., IP routers, ATM switches and Ethernet switches) [1][2][3][4][5]. One well known architecture is the output-queued (OQ) switch which has the following property: When a packet[1] arrives, it is immediately placed in a queue that is dedicated to its outgoing port, where it will wait its turn to depart.

OQ switches have a number of appealing performance characteristics [6]. First, they are work-conserving, which means that OQ switches provide 100% throughput for all types of traffic. They minimize average queueing delay, and — with the correct scheduling algorithms — can provide delay guarantees and different qualities of service. In fact, almost all of the techniques known for providing delay guarantees [7][8][9][10][11] assume that the switch is output queued. Because of its good performance, and simple operation, several switch prototypes have emulated OQ switches using a shared medium [12][13][14], or a centralized shared memory [15][16]. And several switch architectures proposed in the literature attempt to emulate the behavior of an OQ switch, for example Combined Input and Output Queued (CIOQ) switches [17][31], and Parallel Packet Switches (PPS) [18][19].

While each of the above techniques have their pros and cons, the shared memory architecture [20] is the simplest technique for building an OQ switch. Early examples of commercial implementations of shared memory switches include the SBMS switching element from Hitachi [21], the RACE chipset from Siemens [22], the SMBS chipset from NTT [23], the PRELUDE switch from CNET [24][25], and the ISE chipset by Alcatel [26].

A shared memory switch is characterized as follows: When packets arrive at different input ports of the switch, they are written into a centralized shared buffer memory. When the time arrives for these packets to depart, they are read from this shared buffer memory and sent to the egress line.

Some recent examples of commercial shared memory switches are the M40 backbone router from Juniper [27] and the ATMS2000 chipset from MMC Networks [28]. In general, though, there have been fewer commercially available shared memory switches than those that use other architectures. This is because it is difficult to scale the capacity of shared memory switches to the aggregate capacity required today.

A. Why is it considered difficult to scale the capacity of shared memory switches?

Shared memory switches have the following characteristics that make it difficult to scale their capacity.

1. As the line rate R increases, the memory bandwidth[2] of the switch's shared memory must increase. For example, if a packet switch with N ports buffers packets in a single shared memory, then it requires a memory bandwidth of $2NR$.

[1] We shall use the terms packets and cells inter-changeably throughout the this paper.

* This work was funded by NSF ANI-9872761-001 and the Industrial Technology Research Institute (Taiwan).

Thus the memory bandwidth scales linearly with the line rate.

2. If we assume that arriving packets are split into fixed sized cells[3] of size C, then the shared memory would have to be accessed every $A_t = C/2NR$ seconds. For example, if $C = 64$ bytes, $N = 32$ and $R = 10Gb/s$ then the access time is just 800ps, well below the access time of SRAM (static RAM) devices, and two orders of magnitude smaller than the access time of DRAM (dynamic RAM) devices today.

3. Packet switches with faster line rates require a large shared memory. As a rule-of-thumb, the buffers in a packet switch are sized to hold approximately $RTT \times R$ bits of data (where RTT is the round trip time for flows passing through the packet switch), for those occasions when the packet switch is the bottleneck for TCP flows passing through it. If we assume an Internet RTT of approximately 0.25 seconds, a 10Gb/s interface requires 2.5Gbits of memory. If the switch had 32 ports, it would require 10Gbytes of shared memory which, although possible, would be impractical with SRAM today.[4]

In summary, it appears that although it has attractive performance, the shared memory architecture is not scalable because of memory size and memory access time. Hence, centralized shared memory switches have not received much attention in recent years, with research work being focussed instead on input queued [29][30], and CIOQ switches [17][31]. It is the purpose of this paper to revisit the problem of designing high capacity shared memory OQ switches, and explore ways to make them more scalable.

In the next section we give a brief description of several different approaches to building shared memory switches, and introduce the particular approach we explore in the remainder of this paper.

II. Background

It seems that neither of the currently widely available memory technologies (DRAMs and SRAMs) are well-suited for use in large shared memory switches. DRAMs offer large capacity to hold many packets, but their random access time is too slow. On the other hand, SRAMs are fast and might be able to keep up with line rates, but are too small to be economically viable for large packet buffers.

So we'll consider techniques that use a combination of SRAM and DRAM, in the hope that we can find ways to combine the advantages of both, without being limited by their disadvantages. We'll also consider techniques that use a large number of DRAMs arranged in parallel to achieve the memory bandwidth requirements. The different techniques can be broadly categorized as follows:

A. SRAM Only:

In this technique, the shared buffer memory consists of multiple (usually on-chip) SRAMs, which is ideal for low capacity switches, but does not have the storage requirements for large capacity switches because of SRAM's low density. For example, with today's $0.13\mu m$ CMOS technology, the largest available SRAM is approximately 16Mbits [32]. If a single 16Mbit memory was used to store 0.25s of data (for TCP to perform well), then the aggregate capacity would be limited to less than 100Mb/s.

B. SRAM and DRAM:

Since the shared buffer memory must be capable of supporting both fast access time as well as have a large capacity, there have been several techniques proposed which use a combination of SRAM and DRAM.

The main idea is to build a memory hierarchy, where the memory bandwidth is increased by reading (writing) *multiple* cells from (to) DRAM memory in parallel. When packets arrive to the switch they are segmented into cells and stored temporarily in an SRAM, waiting their turn to be written into DRAM. At the appropriate time (determined by a memory management algorithm), multiple cells are written into the

---

2. The memory bandwidth is defined here to be the reciprocal of the time taken to write data to, or read data from a random location in memory. For example, a 32-bit wide memory with a 100ns random access time is said to have a memory bandwidth of 320Mb/s.

3. It is common for packets to be segmented into fixed size units prior to storage. This is done for several reasons: (1) Memory can be utilized more efficiently if all buffers are the same size; (2) Switch fabric scheduling algorithms are often made simpler by using fixed size data units and time slots; and (3) Arriving packets may in fact be 53-byte ATM cells. Throughout this paper, we will refer to the fixed size segments as "cells" of size C, even though they need not be equal in size to an ATM cell.

4. At the time of writing, the largest commercially available SRAM device holds 16Mbits [32], which means 5,000 devices would be needed to hold 10Gbytes.

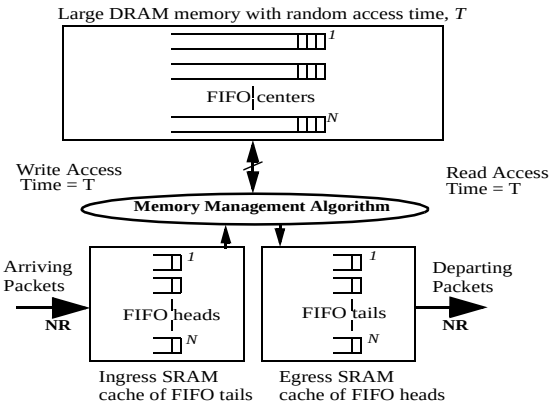

Figure 1: Memory hierarchy of packet buffer, showing both DRAM and SRAM memory.

DRAMs at the same time. We can think of the memory hierarchy as a large DRAM containing a set of FIFOs; the head and tail of each FIFO is cached in a (possibly on-chip) SRAM as shown in Figure 1. A memory management algorithm schedules cells, manages row and bank conflicts,[5] and inter-leaving of memory accesses. The SRAM behaves like a cache, holding packets temporarily when they first arrive and just prior to their departure.

We will consider two performance metrics. First, we'll consider the latency from when a cell is scheduled to depart until it can be retrieved from memory and placed onto the outgoing line. Bear in mind that the scheduling algorithm (for example, a WFQ scheduler) will schedule the cells to depart in an unpredictable order; and most likely not in the same order that they were placed into the shared memory. Ideally, the cell is always available as soon as it is scheduled. In other words, it is already available in the SRAM cache or in some other on-chip storage. As we will see, in some cases, there might instead be a bounded time (statistical bound, or hard bound) from when the cell is scheduled to depart until it is available from memory.

Second, we'll consider the size of the SRAM. A good design will require a smaller SRAM that, ideally, can be placed on-chip. As we will see, the size of the SRAM depends on the types of latency guarantee made. Perhaps not surprisingly, as the guarantees become tighter, we need a larger SRAM.

The approaches that use both SRAM and DRAM can be divided into two techniques.

1. ***Statistical Guarantees:*** In this approach, the memory hierarchy gives statistical guarantees for the latency from when a cell is scheduled until it is ready to be sent on the egress line. This scheme is similar to interleaving or pre-fetching used in computer systems [33][34][35][36][37]. Examples of implementations that use commercially available DRAM controllers are [38][39]. We note however, that unlike computer systems, packet switches cannot tolerate row or bank conflicts that occasionally allow the data to be delivered at an unpredictable time. This means that when statistical guarantees are not met, data might be delivered late, out of order, or not at all.
2. ***Deterministic Guarantees***: An alternative approach for packet switches, is to provide deterministic guarantees. Here, the SRAM is sized so that a cell is always delivered within a bounded delay, regardless of the sequence of traffic patterns. In [40][41] the authors show that if the SRAM is of the order of $O(Q \ln Q)$ cells, (where Q is the number of priority queues maintained in the shared memory switch over all outputs) then cells are always available from memory exactly when needed. On the other hand, the size of the SRAM can be reduced to a size of $O(Q)$ if a bound of $O(Qk)$ time units can be tolerated [42][43][44], where k is the number of DRAM memories used in parallel.

C. DRAM Only:

The following techniques have been used to design shared memory switches using only DRAM:

1. ***Techniques which rely on the DRAM row or bank properties:*** Here, throughput and delays are determined by the probability of row or bank conflicts. A simple technique to obtain high throughputs using DRAMs (using only random accesses) is by striping a cell[6] across multiple DRAMs [45]. In this approach each incoming cell is split into smaller segments and each segment is written into different DRAM banks; these banks reside over a number of parallel DRAMs. With this approach the random access time is still the bottleneck. To decrease the access rate to each DRAM, cell interleaving can be used [46][47]. In this technique, consecutive arriving cells are written into different DRAM banks. However since the order in which packets must depart from the shared memory switch is not known *apriori*, it may happen that consecutive departing

[5] Row and bank conflicts are described in more detail in Section IV.A.

[6] This is sometime referred to as *bit striping*.

cells reside in the same DRAM row or bank, causing row or bank conflicts and momentary loss in throughput.

2. ***Techniques which provide statistical guarantees:*** Here the memory management algorithm is designed so that the probability of DRAM row or bank conflicts is reduced. These include designs that randomly select memory locations [48][49][50], so that the probability of row or bank conflicts in DRAMs are considerably reduced. Under certain conditions, statistical bounds (such as average delay) can be found.

D. Goals of this paper

In the remainder if this paper, we explore whether it is possible to design a shared memory switch using only DRAMs, with deterministic guarantees on the delays faced by a cell. In fact we shall ask a slightly stronger question: *Can we emulate an ideal OQ switch using a shared memory switch in which each DRAM operates slower than the line rate*?

We show in this paper, that this is indeed possible by adapting a memory management technique originally described in [18]. The main result of this paper is that a shared memory switch can emulate a FCFS-OQ switch with each memory operating at $3NR/k$, where N is the number of ports, R is the ingress line rate, and $k > 1$ is the number of DRAMs. The result holds for any packet arrival process. We further prove that a shared memory switch can support multiple qualities of service. To facilitate this we introduce a fixed size buffer of size Nk running at the line rate, and show that the shared memory switch can emulate an OQ switch supporting a variety of QoS disciplines, within a relative delay bound of Nk timeslots, if each memory operates at $4NR/k$.

Access to the DRAMs is controlled by a memory management algorithm that we call CFD-MMA (conflict free DRAM memory management algorithm). By avoiding row and bank conflicts, CFD-MMA enables optimal use of the DRAM bandwidth. We derive the conditions under which CFD-MMA can eliminate DRAM row and bank conflicts.

We find that a shared memory switch having 16 ports and 64 byte cells,[7] each port operating at 40Gb/s (OC768c), can be built from 100 DRAMs each having a random access time[8] of $16ns$.

III. DEFINITIONS

Before proceeding, it will be useful to define some terms used throughout this paper:

Cell — *A fixed-length packet; not necessarily equal in length to a 53-byte ATM cell. Although packets arriving to the shared memory switch may have variable length, we will assume in this paper that they are processed and buffered internally as fixed length "cells". This is common practice in high performance routers; variable length packets are segmented into cells as they arrive, carried across the switch or stored in a shared memory as cells, and reassembled back into packets before they depart.*

Time slot — *Refers to the time taken to transmit or receive a fixed length cell at a link rate of* NR.

Output-Queued (OQ) Switch — *A switch in which arriving packets are placed immediately in queues at the output, where they contend with packets destined to the same output waiting their turn to depart. The departure order may simply be first-come first-served (FCFS) in which case we call it a FCFS-OQ switch.*

In order to compare the performance of a shared memory switch to that of an ideal OQ switch we will need the following four definitions:

Shadow OQ switch — *We will assume that there exists an OQ switch, called the "shadow OQ switch", with the same number of input and output ports as the shared memory switch. The ports on the shadow OQ switch receive identical input traffic patterns and operate at the same line rate as the shared memory switch.*

Mimic — *Two different switches are said to mimic [6] each other if, with identical arrivals, identical packets depart from both switches at the same time.*

Relative queueing delay — *A cell's relative queueing delay is the increased queueing delay (if any) that it receives in a switch relative to the delay it receives in the shadow OQ switch. We define relative queueing delay to only include differences attributed to queueing. Differences in fixed delay (e.g. because of differences in propagation delay, pipelining etc.) are not included in this measure.*

[7] It is common in practice for cells to be 64 bytes long as this is the first power of 2 above the sizes of ATM cells and minimum length TCP segments (40 bytes).

[8] This number represents the random access time when there are no row or bank conflicts.

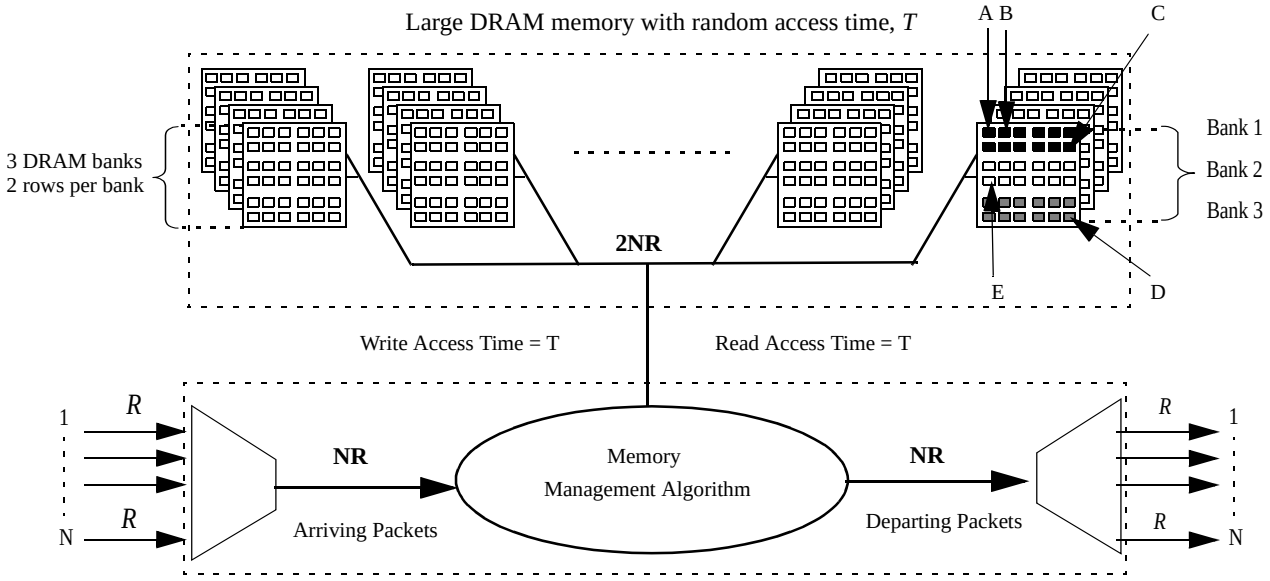

Figure 2: Memory hierarchy of packet buffer, showing large DRAM memory. The DRAM memory has a total bandwidth of at least 2NR. The logical DRAM memory consists of multiple separate DRAMs. In this example, each DRAM is shown having three banks, with each bank having two rows. There are a total of six rows in each DRAM. A, B, C, D, and E denote the names of cells stored in the DRAM.

Emulate *— Two different switches are said to emulate each other, if under identical inputs, they have identical relative queuing delays. Thus the term emulate is identical to mimic if we ignore the propagation delays in the switches. We shall use the term emulate in the rest of the paper to compare the performance of the shared memory switch with an OQ switch.*

IV. The Shared Memory Architecture

In this paper, we shall focus on the specific shared memory architecture shown in Figure 2. The figure shows a shared memory switch with N ports, where each port operates at rate R. We shall assume that cells of size C arrive at a maximum aggregate rate NR. The shared memory switch maintains N FCFS queues, one for each output port. Arriving cells are buffered in a shared buffer memory which consists of k physical DRAMs arranged in parallel. Note that a cell may arrive and depart from the shared memory switch once every C/NR seconds. We denote the access time of the DRAM by T, i.e. the time taken between any two successive memory accesses to the same DRAM. We will assume that a cell of size C can be scheduled to be written to or read from a DRAM every T seconds. A trivial requirement for the shared memory subsystem is that $k \geq 2NRT/C$. We assume that there is a memory management algorithm (CFD-MMA: Conflict Free DRAM Memory Management Algorithm) which determines which cells are written to and read from the DRAMs in every time slot. Before we describe how CFD-MMA operates, we shall clarify some of the characteristic properties of a DRAM.

A. Properties of a DRAM:

A DRAM's memory array is arranged internally in rows, where a set of contiguous rows form a bank. As an example, Figure 2 shows DRAMs organized as 3 banks, where each bank has 2 rows. The access time T of the DRAM (i.e. the time taken between consecutive accesses to any location in the memory array) is dependent on the sequence of memory accesses made to the DRAM. These can be categorized as follows:

1. Consecutive cells in the same row and bank.
   The access time between two consecutive references to adjacent memory locations in the same row of the same bank is denoted by T_C. This is sometimes referred to as the *burst mode* in a DRAM. In Figure 2, consecutive references to cells A and B can be done in burst mode. T_C is usually about 5-10ns today. Although this is fast, it is not common in a packet switch to be able to use burst mode, because successive cells do not usually reside in the same row and bank.

2. Consecutive cells in different rows, but in the same bank.
   The access time between two consecutive references to different rows but belonging to the same bank is denoted by T_{RC}. As an example, consider consecutive references to cells B and C in Figure 2.

We say that there is a *row conflict* because the rows belong to the same bank. T_{RC} represents the worst case access time for random accesses to a DRAM, and it is of the order of 70-80ns for commodity DRAMs available today.

3. Consecutive cells in adjacent banks.
   Some DRAMs such as RDRAM [51], incur a penalty when two consecutive references are made to adjacent banks. i.e. if a cell accesses bank x, then the next cell cannot access banks $x-1$, x or $x+1$. This is called a *bank conflict* and we shall denote it by T_{BC}. As an example, consecutive references to cells D and E in Figure 2 cause a bank conflict.
4. Consecutive cells in different rows.
   The access time for two consecutive references to rows in different banks is denoted by T_{RR} and is called the *row to row* access time. An example in Figure 2 would be a reference to cell A, B or C followed by a reference to cell D. This number is of the order of 20ns and represents the best case random access time for a DRAM.

In a DRAM, there is usually a heavy penalty in the access time for both row and bank conflicts.[9] Thus $T_{RR} \ll T_{RC}, T_{BC}$. We will initially choose the random access time of the DRAM to be the worst case access time i.e. $T = max\{T_{RR}, T_{RC}, T_{BC}\}$.

B. CFD-MMA:

We now describe CFD-MMA. In every time slot, CFD-MMA performs two operations: It writes an incoming cell to the DRAM and reads a cell from the DRAM and sends it to the external line. Without loss of generality, we will assume that during each time slot, the read operation precedes the write operation. Since the CFD-MMA serves N ports, each of which runs at a rate R, it uses one amongst every N time slots to read a cell from each port.

CFD-MMA has three main characteristics. First, it exploits the characteristics of the FCFS shared memory switch to interleave memory accesses. Second, it uses speedup (which we will define shortly) to prevent memory conflicts. Third, it exploits the properties of the DRAM architecture to avoid row and bank conflicts.

Part I: Interleaving memory accesses to avoid busy memories

The DRAMs run k times slower as compared to the aggregate line rate NR, which means that it takes several time slots to write a cell to or read a cell from a DRAM. Hence, when a cell arrives, a particular DRAM might be busy reading or writing a cell, while other DRAMs might be idle.

CFD-MMA keeps track of which DRAMs are idle, and writes the arriving cell into an idle DRAM. But for the cell to be able to depart at the correct time, CFD-MMA must pick a DRAM that will also be idle when the cell needs to depart. CFD-MMA takes advantage of the fact that the departure order of cells is known (as is the case for an FCFS-OQ switch). CFD-MMA determines the set of DRAM(s) which are idle when the cell arrives, and will also be idle when the cell departs. It then writes the cell into one of these DRAMs.

Part II: Using speedup to avoid busy memories.

How do we know that there is a DRAM that is idle when the cell arrives and will be idle when it departs? It turns out that we can't, unless we use speedup. We say that there is *speedup* if the system has more DRAMs than we strictly need. In other words, $k = SNRT/C$. We define the speedup as the ratio between the memory bandwidth required in our system and the minimal bandwidth, $2NR$, required to maintain throughput in any shared memory switch.[10] Hence the speedup required in a shared memory switch is $S/2$. The access time of the DRAM, T is then simply $\lceil k/S \rceil$ time slots. Shortly, we will determine the minimum speedup required to guarantee that there is always at least one DRAM that is idle when a new cell arrives, and will be idle when it departs.

Part III: Exploiting the DRAM architecture to avoid row and bank conflicts.

It is possible for CFD-MMA to use the memory in a way that row and bank conflicts don't arise. To prevent row or bank conflicts we will exploit the fact that each DRAM has multiple banks. We will show that CFD-MMA bounds the number of accesses to a given DRAM in any time interval. If there are more banks than the number of accesses, then CFD-MMA is always guaranteed to find a bank which avoids bank conflicts.

In the next section, we shall formally prove the conditions under which CFD-MMA can emulate a FCFS-OQ switch.

9. There can be other penalties such as the *read-write* turnaround associated with accesses to a DRAM. We do not take into consideration these penalties in this paper.

10. Recall that we need at least $k = 2NRT/C$ DRAMs just to meet the read and write requirements at an aggregate line rate of NR.

V. EMULATING A FCFS-OQ SWITCH

Before we state the theorem, we will need the following definitions.

A. Definitions.

Definition 1: ***Busy Write Set (BWS)-*** *When a cell is written into a DRAM, it is busy for $\lceil k/S \rceil$ time slots. $BWS(t)$ is the set of DRAMs which are busy due to a cell being written at time t, and therefore cannot accept a new cell. Thus, $BWS(t)$ is the set of DRAMs to which the CFD-MMA has initiated a write operation in the previous $\lceil k/S \rceil - 1$ time slots. Clearly $|BWS(t)| \leq \lceil k/S \rceil - 1$.*

Definition 2: ***Busy Read Set (BRS)-*** *Likewise, the $BRS(t)$ is the set of DRAMs which are busy reading cells at time t. It is the set of DRAMs from which the CFD-MMA has initiated a read operation in the previous $\lceil k/S \rceil - 1$ time slots. Clearly $|BRS(t)| \leq \lceil k/S \rceil - 1$.*

Definition 3: ***Departure Time****- When a cell arrives, the CFD-MMA determines its departure time. A cell arriving at time slot t and destined to output j is assigned the departure time $DT(t, j)$. Given that we are assuming the departure order to be FCFS, the departure time is the first time that output j is free and able to send the cell. The departure time for the cell in the shared memory switch is the same as that assigned to the cell in the shadow OQ switch.*

B. The conditions for emulating an FCFS-OQ switch

Theorem 1: (Sufficiency) A speedup of 1.5 is sufficient for CFD-MMA to be able to read and write cells from the DRAM without memory conflicts for an FCFS-OQ switch.

Proof: Consider a cell C that arrives to the shared memory switch at time slot t and destined for output port j. The CFD-MMA will determine C's departure time, $DT(t, j)$. CFD-MMA's choice of a DRAM l to write C into must meet these constraints:

1. DRAM l must not be busy writing or reading a cell at time t. Hence $l \notin BWS(t)$, and $l \notin BRS(t)$.

2. CFD-MMA must also pick a DRAM that is not busy when the cell departs from the switch at $DT(t, j)$: DRAM l must not be busy reading another cell when C is ready to depart: i.e. $l \notin BRS(DT(t, j))$.

Hence our choice of DRAM l must meet the following constraints:

$$l \notin BWS(t) \wedge l \notin BRS(t) \wedge l \notin BRS(DT(t, j)) \quad (1)$$

A sufficient condition to satisfy Equation (1) is:

$$k - |BWS(t)| - |BRS(t)| - |BRS(DT(t, j))| > 0 \quad (2)$$

From Definitions 1 and 2, we know that Equation (2) is true if: $k - 3(\lceil k/S \rceil - 1) - 1 > 0$ i.e. $S \geq 3$. By definition, this corresponds to a speedup of 1.5. ❐

Remark: It is possible that an arriving cell must depart before it can be written to the DRAM i.e. $DT(t, j) < t + T$. In that case the cell may be immediately transferred to the output port j, bypassing the shared memory buffer.

Theorem 2: (Sufficiency) A shared memory switch can emulate a FCFS-OQ switch with a speedup of at least 1.5, where $T = max\{T_{RR}, T_{RC}, T_{BC}\}$.

Proof: A cell which arrives to the shared memory switch, is first given a departure time which is equal to its departure time $DT(t, j)$ on the shadow OQ switch. From Theorem 1 we see that if $S \geq 3$, then CFD-MMA ensures that each cell is written into memory immediately at time slot t and can be read from the DRAM at time slot $DT(t, j)$ without any memory conflicts. If we choose the random access time of the DRAM to be $T = max\{T_{RR}, T_{RC}, T_{BC}\}$, then the only delay that any cell faces as compared to the shadow OQ switch, is the latency of reading and writing from the slower speed DRAMs. The cell as such incurs no relative queueing delay as compared to the shadow OQ switch. Hence, a shared memory switch can emulate an FCFS-OQ switch with $S \geq 3$, which by definition corresponds to a speedup of 1.5 ❐

Remark: Note that there exists a tighter bound if we solve exactly for the values of the floor and ceiling. This is of theoretical interest only when k (and hence the switch capacity) is small.

VI. SUPPORTING QOS IN A SHARED MEMORY SWITCH

In this paper, we shall consider only a specific class of QoS policies as defined below.

A. Definitions.

PIFO Queues. A "Push-In First-Out" [17] queue is defined

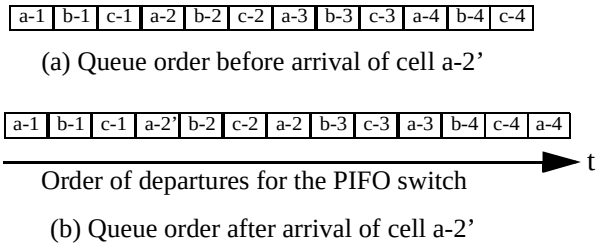

Figure 3: An arriving cell a-2' is pushed in between cells a-1 and a-2 destined for output a. The departure order for the whole switch after insertion is no longer PIFO despite each output having a PIFO scheduling policy.

as follows:

1. Arriving cells are placed at (or, "push-in" to) an arbitrary location in the queue,

2. The relative ordering of cells in the queue does not change once cells are in the queue, i.e. cells in the queue cannot switch places, and

3. Cells may be selected to depart from the queue only from the head of line.

PIFO queues can be used to implement a variety of QoS scheduling disciplines such as WFQ, GPS and strict priorities.

B. Why is a PIFO schedule a problem?

Consider a 3 port shared memory switch, with the output ports named as a, b, and c. As notation, for each cell, we shall designate both the destination port and enumerate the departure order of the cell for that output. Thus a cell b-3 is a cell destined to output port b and is scheduled to be the third cell to depart. similarly, cells b-2 and b-1 denote cells which are also destined to output port b, but which leave later earlier than b-3. An example of such a schedule is shown in Figure 3.a.

Now assume that a new cell arrives for output port a. Since, each output port follows a PIFO scheduling policy, assume that this new cell is inserted after cell a-1 but before cell a-2. We shall call this cell a-2'. When cell a-2' is inserted, it delays the departure time of other cells behind it in output port a. The new departure order is shown in figure Figure 3.b.

We can see that the departure order for the queue maintained by the switch for all outputs, is not PIFO. Now consider any cell destined to output port a, which is queued to depart later than cell a-2'. In particular consider cell a-3. Cell a-3 was written into a DRAM such that it did not conflict with the DRAMs in which cells b-3 and c-3 were stored, as shown in Figure 3.a. However, as per the new departure order in Figure 3.b, cell a-3 departs in a time slot adjacent to the departure time of cells b-4 and c-4. However since cells a-3, b-4, and c-4 have already been written apriori, there is no way to prevent memory conflicts between these cells. This creates a problem, since CFG-MMA, requires cells to be conflict free once they are written to the DRAMs.

C. Creating a PIFO order for the whole switch.

We saw in the previous section that a PIFO insertion done by a single output, can cause it to conflict with cells from other outputs. In order that outputs do not conflict with each other, we do the following. We reorder the output port schedule as follows:-

Output Port Conflict Free Schedule: We create a schedule in which the accesses to different output ports don't clash with each other in a shared memory switch with k DRAMs, by changing the way in which cells are scheduled for departure from an output queued switch. Let the schedule of departures in a switch be denoted by Π = (a-1,b-1,...n-1), , (a-2,b-2,...n-2), ...(a-k,b-k,...n-k), i.e. in each sequence of n time slots a cell departs from each output port. Then in the PIFO shared memory switch, we allocate the schedule as Π' = (a-1,a-1,...a-k), , (b-1,b-2,...b-k), ...(n-1,n-2,...n-k)., i.e. exactly k cells are scheduled to each output in k time slots, and each output gets to read from the DRAM in round robin manner.

Note that, since we read up to k cells from each output, we will also need a small buffer (presumably an on-chip SRAM) of size k cells for each output. Hence the buffer is of size Nk cells.

Creating a PIFO Order: We note that, when arriving cells are given a departure time based on the departure schedule Π', these cells do not clash with cells from other outputs. Thus Π', can be considered to be comprised of a time division multiplexed sequence of departures for each output j, Π'_j, where each Π'_j is a PIFO departure order.

D. The conditions for emulating a PIFO-OQ switch

Theorem 3: (Sufficiency) A speedup of 2 is sufficient for CFD-MMA to be able to read and write cells from the DRAM without memory conflicts for a PIFO-OQ switch.

Proof: Consider a cell C that arrives to the shared memory switch at time slot t and destined for output

port j. The CFD-MMA will determine C's departure time, $DT(t, j)$. CFD-MMA's choice of a DRAM l to write C into must meet these constraints:

1. DRAM l must not be busy writing or reading a cell at time t. Hence $l \notin BWS(t)$, and $l \notin BRS(t)$.
2. CFD-MMA must also pick a DRAM that is not busy when the cell departs from the switch at $DT(t, j)$: DRAM l must not be busy reading another cell when C is ready to depart: i.e. $l \notin BRS(DT(t, j))$
3. DRAM l must not conflict with the $\lceil k/S \rceil - 1$ cells scheduled to be read in parallel from the DRAMs for output j in the next time slot reserved for output j. after $DT(t, j)$.
   i.e. $l \notin BRS(DT(t, j) + Nk + \lceil k/S \rceil - 1)$.

Hence our choice of DRAM l must meet the following constraints:

$$\begin{array}{l} BWS(t) \wedge \\ BRS(t) \wedge \\ BRS(DT(t, j)) \wedge \\ BRS(DT(t, j) + Nk + \lceil k/S \rceil - 1) \end{array} \quad (3)$$

A sufficient condition to satisfy Equation (1) is:

$$\begin{array}{l} k - \\ |BWS(t)| - \\ |BRS(t)| - \\ |BRS(DT(t, j))| - \\ |BRS(DT(t, j) + Nk + \lceil k/S \rceil - 1)| > 0. \end{array} \quad (4)$$

From Definitions 1 and 2, we know that Equation (2) is true if: $k - 4(\lceil k/S \rceil - 1) - 1 > 0$ i.e. $S \geq 4$. By definition, this corresponds to a speedup of 2. ❐

Theorem 4: (Sufficiency) A shared memory switch can emulate a PIFO-OQ switch with a speedup of at least 2, *where* $T = max\{T_{RR}, T_{RC}, T_{BC}\}$.

Proof: The proof is similar to Theorem 2. ❐

VII. Eliminating Row and Bank Conflicts

In the previous section we saw how a shared memory switch can emulate a FCFS-OQ switch using the worst case random access time i.e. $T = max\{T_{RR}, T_{RC}, T_{BC}\}$. We can reduce k if we can access the DRAM every $T = min\{T_{RR}, T_{RC}, T_{BC}\} = T_{RR}$ seconds i.e. by avoiding row and bank conflicts. In this section we shall derive the conditions on the DRAM, such that CFD-MMA can eliminate row and bank conflicts.

A. Eliminating row conflicts

First, we will modify CFD-MMA to avoid row conflicts. Assume cell C arrives in time slot t and is destined to output j. Once CFD-MMA has identified a DRAM l to write C into, it examines all the accesses which have been and will be made to DRAM l in time $(t - T_{RC}, t + T_{RC})$. This is shown in Figure 4. (We will later show how the CFD-MMA can derive a bound on the number of row conflicts during this time interval). Similarly CFD-MMA examines all the accesses to DRAM l in time $(DT(t, j) - T_{RC}, DT(t, j) + T_{RC})$. It then picks a bank in DRAM l which is not used in any of these time intervals to avoid row conflicts.

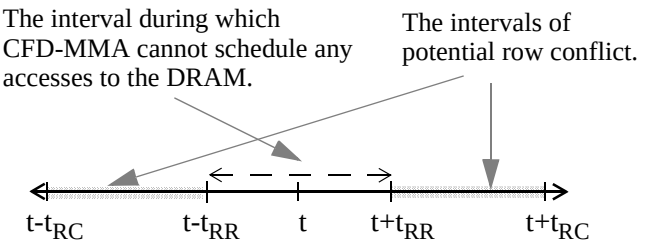

Figure 4: The time intervals during which CFD-MMA has to examine for potential bank conflicts. The shaded regions denote such intervals.

Theorem 5: (Sufficiency) CFD-MMA can completely eliminate row conflicts in a DRAM, if each DRAM has more than $4(\lceil T_{RC}/T_{RR} \rceil - 1)$ *banks.*

Proof: Suppose that a row conflict occurred at time t in DRAM l for cell C destined to output port j. Now, CFD-MMA writes a cell to a DRAM only if it is assured that there are no row conflicts while writing the cell C at time slot t, and that there will be no row conflicts when reading the cell C at its departure time $DT(t, j)$.

Hence if a row conflict occurs, it either occurs or is detected while writing a cell to DRAM. Now again referring to Figure 4, we know that because DRAM l was chosen by CFD-MMA at time t, it could not have been accessed in the time interval $(t - T_{RR}, t + T_{RR})$ or similarly in the time interval $(DT(t, j) - T_{RR}, DT(t, j) + T_{RR})$. The only possible accesses to DRAM l which can cause row conflicts with cell C when it is being written at time t are those in the time intervals: $(t - T_{RC}, t - T_{RR})$ and $(t + T_{RR}, t + T_{RC})$, which are shown shaded in Figure 4. Similarly, the only accesses to DRAM l which can cause a row conflict with cell C when it will be read out at its departure time, $DT(t, j)$, are those in the time intervals $(DT(t, j) - T_{RC}, DT(t, j) - T_{RR})$ and $(DT(t, j) + T_{RR}, DT(t, j) + T_{RC})$. Since, each of these intervals have a length $T_{RC} - T_{RR}$, we know that CFD-MMA will not schedule more than $\lceil (T_{RC} - T_{RR})/T_{RR} \rceil$ accesses for DRAM l in each interval. Hence no more

than $4\lceil (T_{RC} - T_{RR})/T_{RR} \rceil = 4(\lceil T_{RC}/T_{RR} \rceil - 1)$ banks could have been scheduled to be accessed. In order that the cell C be written to DRAM l at time t without any row conflicts, CFD-MMA requires at most one more bank. Hence CFD-MMA can completely eliminate row conflicts, if each DRAM has more than $4(\lceil T_{RC}/T_{RR} \rceil - 1)$ banks. ❐

B. Eliminating bank (and row) conflicts

Bank conflicts occur in different ways in different DRAMs. In general, we are denied access to a bank if we have accessed a nearby bank within the last T_{BC} seconds. We will define W to be the maximum number of banks that can conflict with any given bank in a time slot.[11] As an example, RDRAMs [51] require that if bank x is accessed, the next cell cannot access banks $x-1$, x or $x+1$, and so $W = 3$.

Theorem 6: (Sufficiency) CFD-MMA can completely eliminate row and bank conflicts in a DRAM, if each DRAM has more than $4W(\lceil max(T_{RC}, T_{BC})/T_{RR} \rceil - 1)$ *banks.*

Proof: The analysis is similar to Theorem 5. However, in this case, in order to avoid both row and bank conflicts, CFD-MMA must identify all accesses to the DRAMs in the time intervals, $(t - T_M, t - T_{RR})$, $(t + T_{RR}, t + T_M)$, $(DT(t,j) - T_M, DT(t,j) - T_{RR})$ and $(DT(t,j) + T_{RR}, DT(t,j) + T_M)$, where $T_M = max(T_{RC}, T_{BC})$. From, Theorem 5, we know that CFD-MMA will not schedule more than $4(\lceil max(T_{RC}, T_{BC})/T_{RR} \rceil - 1)$ accesses to any DRAM in the above time intervals. Since, no more than $4(\lceil max(T_{RC}, T_{BC})/T_{RR} \rceil - 1)$ accesses can cause row or bank conflicts, we know, that if the DRAMs have more than $4W(\lceil max(T_{RC}, T_{BC})/T_{RR} \rceil - 1)$ banks, then at least one bank is available in every time slot to prevent bank conflicts. ❐

Theorem 7: (Sufficiency) A shared memory switch, can emulate a FCFS-OQ switch with a speedup of at least 1.5, *where* $T = min\{T_{RR}, T_{RC}, T_{BC}\}$, *without row or bank conflicts if each DRAM has more than* $4W(\lceil max(T_{RC}, T_{BC})/T_{RR} \rceil - 1)$ *banks.*

Proof: This follows from Theorems 2, 5, and 6. ❐

[11] Note that the definition of W includes the bank accessed in the previous time slot; i.e. the bank which causes a row conflict. In that sense a row conflict is a special case of a bank conflict and is subsumed in the definition.

VIII. Discussion

A. Properties and complexity of the shared memory switch

The shared memory switch proposed here requires an aggregate memory bandwidth of $3NR$ in order to emulate an FCFS-OQ switch. It requires $k \geq 3NRT_{RR}/C$ separate memories, where each memory operates slower than the line-rate and can be built from existing DRAMs. The memory management algorithm, CFD-MMA appears simple to implement and — assuming that any shared memory switch must maintain at least N queues — can use existing state information for its scheduling decisions. For each output, it needs to keep track of the next time it is idle. This is the departure time of the next cell that arrives for that output. CFD-MMA must also keep track of which DRAMs are busy writing cells from the previous $\lceil k/S \rceil - 1$ time slots, and which DRAMs will be busy within $\lceil k/S \rceil - 1$ time slots of a cell's departure.

B. Comparing different FCFS switch architectures

We will now compare the shared memory switch to other switch architectures. Table 1, compares the shared memory switch, with an ideal OQ switch, the IQ switch [29][30], and the CIOQ switch [17][31].

TABLE 1 : Comparison of FCFS switch architectures

	Type of Switch	Switch Fabric	Scheduler Complexity	No. of Mem.	Speed of Mem.	Total Bandwidth	Emulate FCFS-OQ
1	FCFS-OQ	Bus	None	N	(N+1)R	N(N+1)R	-
2	IQ	Crossbar	Complex	N	2R	2NR	No
3	CIOQ	Crossbar	Complex	2N	3R	6NR	Yes
4	Shared Mem.	Bus	Simple	k	3NR/k	3NR	Yes

We see that although the IQ switch requires the least overall bandwidth of $2NR$, it cannot emulate a FCFS-OQ switch. Only the CIOQ switch and the shared memory switch can emulate an FCFS-OQ switch. The shared memory switch has a bus as it's switch fabric, and its scheduling algorithm is much simpler than that for a CIOQ switch. Also, the shared memory switch requires a lower aggregate bandwidth of $4NR$ than a CIOQ switch, which requires a total bandwidth of $6NR$.

IX. Conclusions

In 1987 Karol et al. showed that switches with input queues can have lower throughput than those with out-

put queues [52]. In the following few years, many researchers assumed input queueing was inherently limited to low throughput due to head of line blocking. This assumption changed when it became known that, with virtual output queues and a centralized scheduler, an input queued switch can achieve the same throughput as an output queued switch, with significantly less memory bandwidth. However, switches with input queues require a complicated scheduling algorithm, delay cells by an unpredictable amount and hence cannot provide delay guarantees.

It is widely assumed that a shared memory switch, although providing guaranteed 100% throughput and controllable delay, does not scale to very high capacities. While our results indicate that a shared memory switch requires more overall memory bandwidth, the bandwidth of any individual memory can be made arbitrarily small by using more physical devices.

REFERENCES

[1] J. Turner and N. Yamanaka, “Architectural choices in large scale ATM switches,” *IEICE Trans. Communications*, vol.E81-B, no.2, pp.120-137, Feb. 1998.

[2] H. Ahmadi, W. Denzel, “A survey of modern high performance switching,” *IEEE JSAC*, vol. 7, pp. 1091-1103, Sep. 1989.

[3] R. Y. Awdeh and H. T. Mouftah, “Survey of atm switch architectures,” *Comp. Networks and ISDN Systems*, vol. 27, pp. 1567-1613, 1995.

[4] F. A. Tobagi, “Fast packet switch architectures for broadband integrated services digital networks,” *Proc. of the IEEE*, vol. 78, No. 1, pp. 133-167, Jan 1990.

[5] N. McKeown, “A fast switched backplane for a gigabit switched router," *Business Communications Review*, Vol. 27. No. 12, 1997.

[6] George Kesidis, and Nick McKeown, "Output-buffer ATM packet switching for integrated-services communication networks," *Presented at ICC '97, Montreal, Canada, Aug. 1997.*

[7] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queuing algorithm,” *ACM Computer Communication Review* (*SIGCOMM'89*), pp. 3-12, 1989.

[8] A. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: The Single Node Case,” *IEEE/ACM Transaction on Networking*, Vol. 1, No. 3, pp. 344-357, June 1993.

[9] L. Zhang, “Virtual Clock: A new traffic control algorithm for packet switching networks,” *ACM Transactions on Computer Systems*, vol.9 no.2, pp.101-124, 1990.

[10] J. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing", *In Proc. of IEEE INFOCOM '96*, pp. 120--128, San Francisco, CA, USA, March 1996.

[11] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round robin,” in *Proc. ACM Sigcomm*, Sep 1995, pp. 231-242.

[12] T. Aramaki, H. Suzuki, S. Hayano, T. Takeuchi, “Parallel ‘ATOM’ switch architecture for high speed ATM networks,” *Proc. IEEE Int. Conf. Comm.*, pp. 250-254, 1992.

[13] T. Takeuchi, T. Yamaguchi, H. Niwa, H. Suzuki and S. Hayano. “Synchronous composite packet switching - A switching architecture for broadband ISDN,” *IEEE JSAC*, pp. 1365-1376, 10/87.

[14] H. Suzuki, H. Nagano, T. Suzuki, “Output-buffered switch architecture for asynchronous transfer mode,” *ICC*, June 89.

[15] H. Kuwahara, N. Endo, M. Ogino, T. Kosaki, “A shared buffer memory switch for an ATM exchange,” *Proc. IEEE Int. Conf. on Comm.*, pp. 118-122, 1989.

[16] M. Devault, J. Y. Cochennec, M. Server, “The prelude ATD experiment: assessment and future prospects,” *IEEE JSAC*, vol. 6, pp. 1528-1537, 1988.

[17] S. Chuang, A. Goel, N. McKeown, B. Prabhakar, “Matching output queueing with a combined input/output-queued switch,” *IEEE J.Sel. Areas in Communications*, Vol. 17, no. 6, pp. 1030-1039, June 1999.

[18] S. Iyer, A. Awadallah, N. McKeown, “Analysis of a packet switch with memories running slower than the line rate,” in *Proc. IEEE INFOCOM ‘00*, pp.529-537.

[19] S. Iyer, N. McKeown, “Making parallel packet switches practical,” in *Proc. IEEE INFOCOM ‘01*, vol.3, pp. 1680-1687.

[20] M. Arpaci and J. Copeland, “Buffer management for shared-memory ATM switches,” *IEEE Comm. Surveys and Tutorials*, vol. 3, no. 1, First Quarter 2000.

[21] N. Endo, T. Kozaki, T. Ohuchi, H. Kuwahara, S. Gohara: Shared buffer memory switch for an ATM exchange," *IEEE Transactions on Communications*, vol. 41, no. 1, January 1993, pp237-245

[22] R. H. Hofmann, R. Muller, “A multifunctional high-speed switch element for ATM applications," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 7, July 1992, pp1036-1040

[23] H. Yamada, S.-I. Yamada, H. Kai, T. Takahashi:, “Multi-Purpose memory switch LSI for ATM-based systems," *GLOBECOM '90*, San Diego, California, paper 805.4, pp1602-1608

[24] M. Devault, J.-Y. Cochennec, M. Servel, “The `PRELUDE' ATD Experiment: Assessments and Future Prospects," *IEEE Journal on Selected Areas in Communications*, vol. 6, part 9, December 1988, pp1528-1537

[25] J.-P. Coudreuse, M. Servel, “PRELUDE: An asynchronous time-division switched network," *ICC '87*, June 7-10, Seattle, paper 22.2, pp. 769-772

[26] M. A. Henrion, G. J. Eilenberger, G. H. Petit, P. H. Parmentier, “A multipath self-routing switch," *IEEE Communications Magazine*, April 1993, pp46-52

[27] C. Semeria, “Internet backbone routers and evolving internet design,” available at http://www.juniper.net/techcenter/techpapers/200002.html

[28] MMC Networks. “ATMS2000: 5 Gb/sec switch engine chipset,” available at http://mmcnet.com/Solutions/ATMS2000.asp, 1996.

[29] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M. Horowitz, “The Tiny Tera: A packet switch core", *IEEE Micro*, Jan-Feb 1997, pp 26-33.

[30] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, “High speed switch scheduling for local area networks,” *ACM Transactions on Computer Systems* 11, 4 (November 1993), pp. 319--352

[31] P. Krishna, N. Patel, A. Charny, R. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE J. Sel. Areas in Communications*, Vol. 17, no. 6, pp. 1057-1066, June 1999.

[32] http://www.necel.com

[33] J. Corbal, R. Espasa, and M. Valero, “Command vector memory systems: High performance at low cost,” *In Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pp. 68-77, October 1998.

[34] B. K. Mathew, S. A. McKee, J. B. Carter, and A. Davis, “Design of a parallel vector access unit for SDRAM memory systems,” *In Proceedings of the Sixth International Symposium on High- Performance Computer Architecture*, January 2000.

[35] S. A. McKee and Wm. A. Wulf, “Access ordering and memory-conscious cache utilization,” *In Proceedings of the First International Symposium on High- Performance Computer Architecture*, pp. 253-262, January 1995.

[36] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” *In Proc. of the 27th Annual International Symposium on Computer Architecture*, pp. 128-138, June 2000.

[37] T. Alexander and G. Kedem, “Distributed prefetch-buffer/cache design for high performance memory systems”, *In. Proceedings. of the 2nd International Symposium on High-Performance Computer Architecture*, pp. 254-263, Feb. 1996.

[38] W. Lin, S. Reinhardt, D. Burger, “Reducing DRAM latencies with an integrated memory hierarchy design,” *In Proc. 7th Int. symposium on High-Performance Computer Architecture*, January 2001.

[39] S.I. Hong, S.A. McKee, M.H. Salinas, R.H. Klenke, J.H. Aylor, and Wm.A. Wulf, “Access order and effective bandwidth for streams on a direct rambus memory,” *In Proceedings of the Fifth International Symposium on High- Performance Computer Architecture*, pp. 80-89, January 1999.

[40] H. Gail, G. Grover, R. Guerin, S. Hantler, Z. Rosberg, M. Sidi, “Buffer size requirements under longest queue first,” *Proceedings IFIP'92*, vol. C-5, pp. 413-24, 1992.

[41] S. Iyer, R. R. Kompella, and N. McKeown, “Techniques for fast packet buffers,” *In Proc. GBN Workshop*, Anchorage, Alaska, 2001.

[42] S. Iyer, R. R. Kompella, and N. McKeown, “Analysis of a memory architecture for fast packet buffers,” *In Proc. IEEE HPSR*, Dallas, Texas, 2001.

[43] A. Birman, H.R. Gail, S.L. Hantler and Z. Rosberg, “An optimal service policy for buffer systems,” *Journal of the Association for Computing Machinery*, pp. 641-57, vol. 42, no. 3, May 1995.

[44] G. Sasaki, “Input buffer requirements for round robin polling systems,” *In Proceedings of 27th Annual Conference on Communication, Control and Computing*, pp. 397-406, 1989.

[45] P. Chen and David A. Patterson, “Maximizing performance in a striped disk array”, *ISCA*, pp. 322-331, 1990.

[46] “Doubling memory bandwidth for network buffers”, Youngmi Joo and Nick McKeown, *Proc. IEEE Infocom* 1998, vol. 2, pp. 808-815, San Francisco.

[47] Patterson, D. and J. Hennessy, *Computer Architecture: A quantitative approach*, 2nd. ed., San Francisco: Morgan Kaufmann Publishers, c1996.

[48] L. Carter and W. Wegman, “Universal hash functions,” *Jour. of Computer and System Sciences* 18, 1979, pp. 143-154.

[49] R. Impagliazzo and D. Zuckerman, “How to recycle random bits”, *Proc. of the Thirtieth Annual Symposium on the Foundations of IEEE*, 1989.

[50] B.R. Rau, M.S. Schlansker and D.W.L. Yen, “The Cydra 5 stride-insensitive memory system”, *In Proc. Int Conf. on Parallel Processing*, 1989, pp.242-246.

[51] Rambus Inc., www.rambus.com.

[52] M. Karol, M. Hluchyj, S. Morgan: "Input versus output queueing on a space-division packet switch", *IEEE Trans. on Communications*, vol. COM-35, no. 12, December 1987, pp. 1347-1356.