Scalable Router Memory Architecture Based on Interleaved DRAM

Feng Wang and Mounir Hamdi
Computer Science Department of
The Hong Kong University of Science and Technology
{fwang, hamdi}@cs.ust.hk

Abstract – Routers need buffers to store and forward packets, especially when there is network congestion. With current memory technology, neither the SRAM nor the DRAM *alone* is suitable for high-speed Internet routers which require both large capacity and fast access time. Some previous work has been done to combine the two technologies together and make a hybrid memory system [1]. In this paper[1], we propose another hybrid memory system based on the interleaved DRAM memories. We devise an efficient memory management algorithm to provide hard performance guarantees to the memory system. The main contribution of this architecture is that it can scale to a very large capacity with interleaved DRAM while only employing necessary SRAM of the same size as in [1]. Another advantage of this architecture is that the interleaved DRAM provides flexibilities to make the memory management algorithms efficient and the memory system very responsive at high speed rates.

I. INTRODUCTION

Routers need buffers to store and forward packets, especially when there is network congestion [2]. In today's data communication networks, memory technology plays a very important role in affecting the performance of routers, and as a result affecting the performance of the whole network.

What makes the router memory difficult to build can be elaborated in two dimensions. One is the capacity requirement of the router memory. Take Internet as an example, it is a rule of thumb that the routers should be capable of holding packets at least for packets' round-trip life time. This fact results in a buffer size requirement in routers of $RTT \times R$ [3], where RTT is a round-trip time for a packet and R is the line rate. Assuming an RTT of 250 *ms* and the line rate R of 40 *Gbps*, the buffer size of one single port equals to 10 *Gbits*. The other dimension of the difficulties is the memory access time requirement. Still assuming a line rate R of 40 *Gbps* and the average size of one single packet to be 40 bytes, the router memories are required to be able to accommodate one packet every 8 *ns*.

With current memory technologies, neither of the SRAM or DRAM [4] can meet the two requirements simultaneously. The SRAM is fast enough with an access time of around 4 *ns*, but it cannot be built with large capacities (normal capacity is of tens of MB) and is power-hungry as well. The DRAM can be built with a large capacity, but the access time is too large, which is around 40 *ns*. The reason behind this dilemma is that current memories (architectures) are most optimized for *computer* architecture where instruction/data stored in memories will be *reused* for considerably many times. Hence, the hierarchical cache based memory architecture suits the computer architecture quite well. While in router memories, data (packets) are *seldom* reused. Every packet just comes into the memory and leaves sometime afterwards, normally only *once* and never comes back again. In this situation, the cache techniques widely used in computer memory architecture will find little use in the router context since the time or space locality properties do not hold any more.

What makes the router memory more challenging to build is that we should maintain multiple queues, rather than just one single first-in-first-out (FIFO) queue. Intuitively, dispatching and storing packets in multiple separate queues will cause a lot of overhead to the memory management algorithms.

One natural idea is to *combine* the SRAM and DRAM technologies together and make hybrid memory architectures with both large capacity and a reasonable small access time. Some previous work has been carried out in [1] and [5]. In [1], the authors proposed a memory architecture with one large DRAM memory sandwiched by two smaller SRAM memories, as shown in figure 1. The DRAM serves as the main storage for packets, and the head and tail SRAM serve as two buffers for fast reading and writing. Scheduling packets between the SRAM and the DRAM is what was called a memory management algorithm (MMA). They proposed three memory management algorithms that have different tradeoffs between the size of the SRAM and the pipelined delay. Their main idea is to wait in the SRAM to gather b packets in one queue, and then transfer them together to the DRAM in just *one write operation* of the DRAM. In this way, they hide the access time gap between the SRAM and the DRAM (assuming the access time of the DRAM is b times that of the SRAM). In [5], the memory architecture was nearly the same, except that the authors used b interleaved DRAM memories to replace the one single DRAM used in [1]. Packets from the SRAM were written into the distributed DRAM memories in an interleaved way. In this way, they can match the access time of SRAM and DRAM. However, the memory management between the SRAM and the DRAM was based on some randomized algorithm. So, the performance of this proposal is only guaranteed with a given probability. A packet with an uncertain delay or dropped not because of congestions in the memory systems will cause serious overhead in the whole data networks (e.g. unnecessary TCP retransmissions). Thus, the proposal in [5] will not see much practical use in real systems.

In this paper, we propose a memory architecture bearing two aims in mind. *First*, the memory system should scale to large capacities. *Secondly*, the memory system should be fast

[1] This work is supported under RGC HKUST6260/04E.

responsive to burst traffic. We also use interleaved DRAM in the middle for main storage as that in [5], but we use a different memory management algorithm. We use a deterministic memory management algorithm to schedule packets between the head/tail SRAM and the interleaved DRAM. Thus, we can make hard delay and throughput guarantees, rather than probabilistic ones. On the other hand, we can provide better average performance than that in the architecture in [1].

The rest of this paper is organized as follows. In section II, we introduce currently proposed hybrid SRAM/DRAM network memory architectures. Then in section III, we propose our architecture and devise a memory management algorithm (MMA) for it, which is based on a new model of a bipartite matching problem. We analyze the performance of the MMA in section IV and discuss some practical considerations in section V. Then, we conclude the paper.

II. Related Work

Building memories for high performance routers has been a challenging task for a long time. We are not sure about how companies make their router memory products (typically is classified), and there seems to be just a little work in academia talking about this issue.

The most concrete and practical result came from [1]. The authors used hybrid memory architecture as shown in figure 1. Logically, the hybrid memory maintains a set of Q first-come-first-out (FIFO) queues. The head and tail of each FIFO queue reside in the SRAM, while the middle of the FIFO queue resides in the DRAM. As packets arrive at the memory system, they are immediately written to the tail of an FIFO queue in the SRAM cache, where they wait for the memory management algorithm (MMA) to transfer them to the corresponding FIFO queue in the DRAM. The MMA always transfers a block of b packets every time, *never smaller*, from an SRAM FIFO queue to the corresponding DRAM FIFO queue. Similarly, the MMA always transfers packets from the DRAM queues to the SRAM queues in blocks of b packets. Note that b is normally the ratio of the access time of the DRAM to that of the SRAM. Different write granularities to the SRAM and the DRAM (one write per packet to the SRAM and one write per b packets to the DRAM) hide the access time gap between them.

Two main performance *metrics* of this architecture are the SRAM size needed and the packet delay, if any, under the MMA. We prefer smaller SRAM size since the cost of the SRAM is normally very high and smaller SRAM might be built on-chip as well. The packet delay dictates the promptness of fulfilling the arbiter requests. The SRAM is sized so that whenever the arbiter requests a packet, it is always delivered within a bounded delay, regardless of the sequence of requests. Due to the symmetric property of the architecture in figure 1, it suffices to focus on only packets shuttling between the DRAM and the head SRAM, which is how to replenish the head SRAM to prevent a packet request loss from the outside.

Intuitively, since the MMA can only shuttle a block of b packets (never smaller, and this is determined by the architecture) of the *same* queue from the DRAM to the SRAM, the size of SRAM should be large enough for holding some pre-fetched packets that are not requested by the arbiter, but will be used later. The MMA employed in [1] are based on a *look-ahead scheme* to replenish the FIFO queues in the SRAM. The MMA looks ahead sufficient many packet requests from the arbiter. Then it combines this information with the number of present packets in the head SRAM FIFO queues and determines which queue will become the first to be empty under the sequence of arbiter requests. That queue is named as the earliest *critical* queue under the current request pattern. The MMA chooses that critical queue and replenishes it with packets from the corresponding DRAM FIFO queue. The MMA is called the earliest critical queue first (ECQF) MMA. In particular, the authors stated that a head SRAM buffer size of $Q(b-1)$ packets and a look-ahead of $Q(b-1)+1$ packets are sufficient for the ECQF-MMA to service any sequence of arbiter requests with a latency of $Q(b-1)+1$ time slots. The *time slot* is defined as the smallest time gap between two consecutive requests.

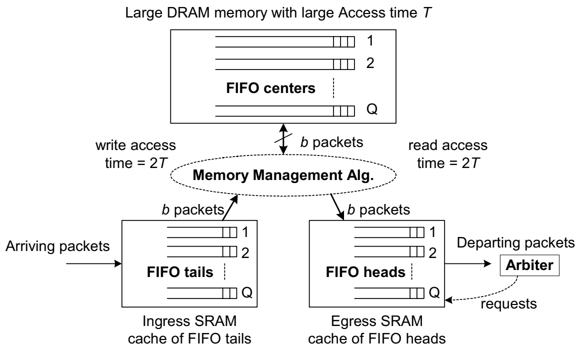


Figure 1: Memory hierarchy of packet buffer, showing DRAM memory with heads and tails of each FIFO maintained in a smaller SRAM cache [1].

One of the constraints of their scheduling algorithm is the b packets as a block to be shuttled between the SRAM and the DRAM. It is essential for the MMA to *hide* the access time gap between the SRAM and the DRAM.

Another proposed architecture is in [5]. They use interleaved DRAM instead of one single DRAM. The MMA in [5] *hides* the access time gap between the SRAM and the DRAM by dispatching write (read) b packets into b interleaved DRAM. It is obvious to see that the packet-scheduling algorithm here is expected to be more complex since the MMA is dealing with packets in a finer granularity, one packet per DRAM. Interleaved DRAM can be viewed as distributed resources, and the simplest way to coordinate distributed resources is to employ probabilistic algorithms. In fact, in [5] the authors have given a simple MMA based on *uniformly randomly* dispatching packets into the b DRAM simultaneously. They stated that under certain conditions, chances of packet conflicts would be very small. However, in router memories packet drops due to reasons other than memory overflow (an indication of network congestion) are always undesirable since this will cause too many unnecessary cross-layer adjustments (Take TCP retransmission as an example).

III. Hybrid Architecture with Interleaved DRAM

We build our hybrid memory architecture based on two observations on the memory architecture [1] mentioned above.

First, always transferring a block of *b* packets between the SRAM and the DRAM certainly makes the MMA simple, but waiting for collecting *b* packets of an FIFO queue surely add some unnecessary delays for some packets, especially in bursty traffic conditions.

Second, [1] cannot use more than one single DRAM in the middle for main packet storage. This is determined by its MMA (always transferring *b* packets, never less and never more). The whole memory system is still limited within a few *Gbits* with current DRAM technology. Using more than one DRAM and making them interleaved will surely increase the size, but it needs a new memory management algorithm.

Although [5] tried to use interleaved DRAM to increase the whole memory capacity, their MMA is based on probability guarantees.

In this paper, we employ the interleaved DRAM architecture to scale the memory system. In particular, we devise an efficient *deterministic* MMA to break the '*block of b packet*' constraint.

A. Hybrid Memory Architecture with Interleaved DRAM

Figure 2 demonstrates our hybrid memory architecture with the SRAM in the head and tail and interleaved DRAM in the center containing the majority of packets. The only difference between figures 1 and 2 is in the center DRAM memories. Figure 2 uses *b* interleaved DRAM memories to replace the single DRAM in figure 1.

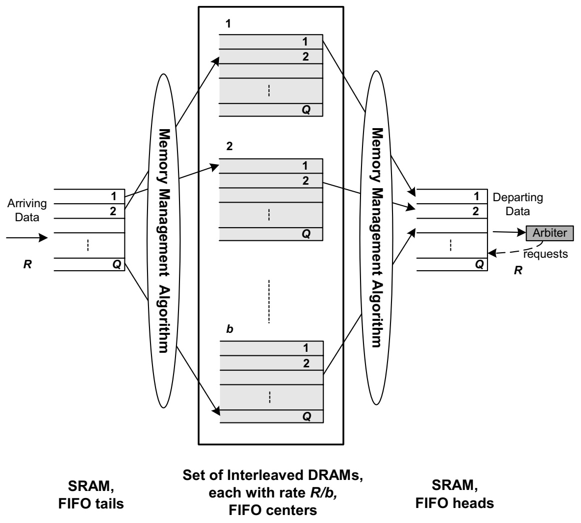

Arriving Data
R
1
2
Q
Memory Management Algorithm
1
2
b
Memory Management Algorithm
Departing Data
Arbiter
requests
R
SRAM, FIFO tails
Set of Interleaved DRAMs, each with rate R/b, FIFO centers
SRAM, FIFO heads

Figure 2: Hybrid memory architecture with interleaved DRAM in the center and smaller SRAM in the heads and tails of each FIFO queue. The MMA resolves the write/read conflicts of individual DRAM.

We define parameters in figure 2 here. Logically, the memory system maintains *Q* FIFO queues from the outside point of view. We interleave *b* DRAM memories in our basic architecture, where *b* is the ratio of the access time of the DRAM to that of the SRAM. The value of *b* is normally around 15 according to current memory technology. We will see later that we can even make *b* larger than this ratio so that we can make the memory system more scalable. For simplicity of the presentation, we define the access time of SRAM *one time slot*. We also assume, in one time slot, one packet comes in and the arbiter issues one request. We can see that the access time of the DRAM is *b* time slots. That is to say, in *b* time slots, the DRAM can be written (read) once.

In figure 1, the MMA transfers *b* packets of the same FIFO queue as a whole into the DRAM in *b* time slots. In figure 2, the MMA transfers *b* packets (not necessarily from the same queue) from the SRAM to feed the *b* interleaved DRAM memories individually (one packet per DRAM) in *b* time slots. We maintain *Q* FIFO queues in every central DRAM and the head and tail SRAM memories.

We make two assumptions here. First, we assume all incoming packets of the same size. Second, we assume we have heavy traffic conditions. That is to say, there are always packets arriving at the tail SRAM and sufficiently many packets are present in the DRAM. If the traffic is light, packets may not go through the DRAM memories. We defer this discussion later.

B. Packet Placement in the Interleaved DRAM Memories

Packets belonging to the same FIFO queue will be placed into the central DRAM memories in a *round robin* way, as show in figure 3. But they are not necessarily transferred into the DRAM in their arriving order. In particular, when a packet comes into the SRAM, the DRAM which it should be written into is determined immediately, but when to write the packet to it is determined by the MMA. This provides flexibility to the MMA. Assuming the packet is the *i*-th packet in its FIFO queue, it should be written into the *j*-th DRAM, where $j = i \bmod b$, as shown in figure 3.

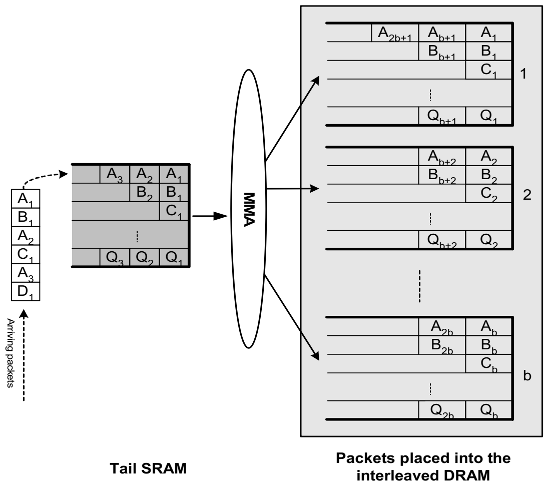


Figure 3: Packets belonging to the same queue are placed into the interleaved DRAM memories in a round robin fashion. The MMA is designed to schedule packets from the tail SRAM to DRAM memories without write conflicts.

In this paper, we only study scheduling packets between the tail SRAM and the central interleaved DRAM, which corresponds to the left MMA in figure 2. It is interesting to notice that scheduling packets between the interleaved DRAM and the head SRAM which corresponds to the right MMA in figure 2 is similar to the left MMA due to the symmetric property of this architecture. Put in another way, we can queue the requests from the arbitrary arbiter in the head FIFO queues and think of them as negative packets. Scheduling packets from the central DRAM to the head SRAM can be thought of as scheduling the negative packets (requests) from the head SRAM to the central DRAM.

IV. Scheduling the Interleaved DRAM

The MMA in figure 1 mainly determines which queue in the SRAM should collect b packets in a block and then transfer them into the DRAM. The MMA in figure 2 mainly determines which b packets should be transferred into the b individual DRAM memories without *conflict*. We say there is a *conflict* in a DRAM, if two packets are required to be written into that DRAM in b time slots. It is obvious to see that b consecutive packets belonging to the same queue surely have no conflict, since they are distributed into the DRAM in a round robin way.

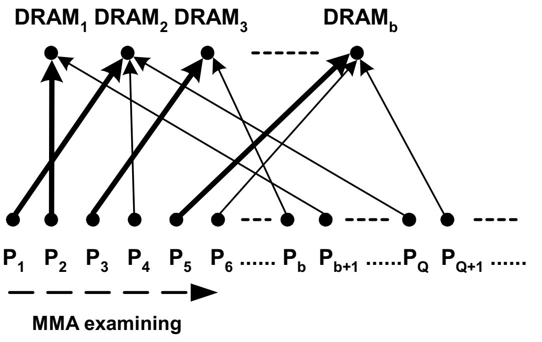
However, always scheduling '*b packets of the same queue*' will surely add some unnecessary delay to some packets. We can make the scheduling granularity finer by breaking the '*b packets of one queue*' constraint and try to transfer packets upon arriving at the tail SRAM into the DRAM as soon as possible. Since every packet has a definite DRAM to be written into, write conflicts may occur between two consecutive packets. The MMA is mainly to resolve this kind of conflicts.

For the simplicity of presentation, we assume one single FIFO queue in the tail SRAM. All packets belonging to different queues just queue in one single queue ordered by their arriving time, as shown in the vertical queue in figure 3.

The MMA runs in *rounds*. In general, in one round b packets are transfers from the tail SRAM to the b interleaved DRAM memories (one packet per DRAM). One round takes b time slots since the access time of each DRAM is b time slots.

In the beginning of one round, the MMA examines packets from the head of the SRAM FIFO queue. Every packet knows exactly which DRAM it should go into. If the DRAM the packet will go into is not reserved, the packet *reserves* that DRAM. Otherwise, the MMA just skips this packet and examines the next packet in the SRAM FIFO queue. The MMA stops examining packets when all b DRAM memories are reserved and then transfers all the b packets that reserve the DRAM memories. Then the MMA cancels all the DRAM reservations and starts the next round of examination and transferring. In fact, examining and transferring packets can be pipelined.

We formulate this algorithm as a Cumulative Matching (CM) problem in the bipartite graph and call the corresponding memory management algorithm CM-MMA.



$DRAM_1$ $DRAM_2$ $DRAM_3$ $DRAM_b$
P_1 P_2 P_3 P_4 P_5 P_6 ...... P_b P_{b+1} ......P_Q P_{Q+1} ......
MMA examining

Figure 4: Bipartite graph for Cumulative Matching

In figure 4, the upper row of b vertices stand for the b DRAM memories and the lower row of infinite vertices stand for incoming packets. An arrow edge from a packet to a DRAM means that the packet wants to reserve that DRAM. The CM-MMA examines packets from the left to the right and makes reservations. One successful reservation corresponds to one matching (the thick line) in the bipartite graph in figure 4. When the b DRAM memories are all reserved, we say the *maximum* matching has been found and the CM-MMA starts to transfer the matched packets from the SRAM to the DRAM memories. Then the CM-MMA removes the corresponding b vertices in the lower row and starts examining packets again from the left for the next round of matching.

We note here that Q is much larger than b. In fact, in practice Q is at the order of thousands. Intuitively, since the value of b is around 15, the CM-MMA will not advance right too much to find a maximum matching.

V. Analysis of the CM-MMA

Now we will investigate the properties of the cumulative matching in figure 4.

It is obvious to see that the CM-MMA guarantees that there is no starvation of any packet since the head packet of the SRAM FIFO queue is always matched in the final matching. That is to say, the CM-MMA advances at least one packet forward in one round.

Lemma 1: Under the CM-MMA, the maximum matching is always achievable if the number of packets in the SRAM is larger than $Q(b-1)$.

Proof: (By contradiction)

We start the proof from the very first round of the CM-MMA when packets belonging to the same queue are assigned orders consecutively. Assume that the CM-MMA has examined $Q(b-1)+1$ packets and there is at least one DRAM remaining unmatched. This means that none of the $Q(b-1)+1$ packets reserves that DRAM. Since packets belonging to the same queue are assigned to the DRAM memories in a round robin fashion and they are consecutively ordered, no queue can contain more than $b-1$ packets. Otherwise, the DRAM will be reserved by at least one of the packets from that queue. Since there are Q queues in the system, packets in the SRAM should not exceed $Q(b-1)$, which leads to a contradiction.

For the following rounds of matching, packets belonging to the same queue may not be consecutive since some packets are removed from the queue according to the CM-MMA, but we can still prove that every DRAM will be reserved if the number of packets in the SRAM is larger than $Q(b-1)$. The intuition here is that in every round b packets are switched out and b packets are filled in the SRAM as well. We defer the detailed proof to the appendix. ■

Lemma 2: Under the CM-MMA, given a DRAM j, for any $Q(b-1)+1$ packets in the SRAM, there are at most $\frac{2Q(b-1)+1}{b}$ packets that reserve the DRAM j.

This lemma is also easy to understand. For the very first round of matching, since every two packets from the same queue reserving the same DRAM will be jammed with at least other $b-1$ packets from the same queue. That is to say, packets reserve the same DRAM cannot be too close. However, for the following rounds of matching, packets of the same queue will be non-consecutive. However, this property still holds for that situation. We defer the detailed proof to the appendix. ■

In summary, the two lemmas state this *property*: given a DRAM j, under the CM-MMA, for $Q(b-1)+1$ packets residing in the SRAM, there are at least *one* packet and at most $\frac{2Q(b-1)+1}{b}$ packets that reserve the DRAM j.

We say an MMA is *stable* if with this MMA the SRAM will not overflow under any arriving traffic patterns.

Theorem: A size of $Q(b-1)+1$ for the SRAM suffices for the CM-MMA to be stable in transferring packets from the SRAM to the interleaved DRAM memories. The possible delay of one packet is bounded within $2Q(b-1)+1$ time slots.

Proof: From lemma 1, we can see that if the size of the SRAM is $Q(b-1)+1$ and when the SRAM is full with packets, the CM-MMA will always find a maximum matching and dispatch the b packets into b DRAM memories in b time slots. In the meanwhile, there are at most b packets arriving at the SRAM in the b time slots. So, the number of packets in the SRAM will never exceed $Q(b-1)+1$. This means that the CM MMA is stable.

For the delay analysis, consider a packet P and it servers DRAM j. Lemma 2 states that in the SRAM there are no more than $\frac{2Q(b-1)+1}{b}$ packets which reserve the DRAM j if the size of the SRAM is $Q(b-1)+1$. This suggests that packet P will be matched no later than the $\frac{2Q(b-1)+1}{b}$-th round, which leads to a delay of no more than $2Q(b-1)+1$ time slots since the CM-MMA takes b time slots in one round in dispatching packets from the SRAM to the interleaved DRAM. ■

VI. Practical Considerations

We note here that the size of the SRAM requirement is the same as that in [1], but we use the interleaved DRAM in the center which can scale to a very larger size. That is to say, we use the same small size of the SRAM to support a lager DRAM main storage.

To make the interleaved memory architecture even more scalable, we can extend b to a larger value, say, $2b$. It is interesting to see that even if we use $2b$ DRAM interleaved, we can still define the maximum matching to be b matches and only dispatch b packets into the $2b$ DRAM since the access time of the DRAM is b and b matches are enough to make the CM-MMA stable. It is easy to see that the size of the SRAM and the delay of packets do not scale with the expansion of the interleaved DRAM. It is only dictated by the ratio of the access time of the DRAM to the SRAM and the number of queues the memory system should maintain.

The worst case packet delay of the CM-MMA is competitive to that in [1]. However, since we break the '*b packets of the same queue*' constraint, the performance is expected to be better, especially under some light traffic conditions for all queues. Assuming a light traffic condition and there are just several (say, less than b) packets in every queue, the MMA in [1] will keep waiting until one of the queues contains more than b packets, while the CM-MMA may still find a maximum matching since the bunch of b packets here do not necessarily come from the same queue. What is more, note that b packets from the same queue naturally form a maximum matching in CM-MMA. Therefore, the CM-MMA is responsively faster than the MMA in [1]. Put in another way, the MMA in [1] transfers packets in a granularity of b packets from the same queue, while the CM MMA transfers packets in a granularity of single packets and combines them to be a bunch of b packets. We will have numerical analysis and simulation results in a sequel paper. With real implementation that we state below, we can further improve the packet promptness, thus decreasing the packet delays.

Both advantages we state above seem to come at the cost of the high-complexity CM-MMA. It is true that the crude version CM-MMA has a time complexity of $O(Q)$ since we may have to examine at most $Q(b-1)+1$ packets before finding a maximum matching (b can be viewed as a constant). However, we can amortize this complexity to each packet to make the time complexity to be $O(1)$. We can notice these two facts: a) packets do not change their reservations even if they fail in one round, and b) a previous reservation has priority over the later reservation to the same DRAM. Using these two facts, in real implementation we can buffer the reservation requests in every DRAM. Moreover, the transmission of matched packets does not need to be synchronized either. In particular, we can transfer a packet immediately after its arrival to the buffer of the corresponding DRAM. In this way, we do not need the $Q(b-1)+1$ size SRAM in the tail any longer, but we need SRAM buffers in each DRAM. Using lemma 2, it is obvious to see that the size of one buffer in the DRAM is less than $\frac{2Q(b-1)+1}{b}$. Therefore, the total size of the SRAM is $2Q(b-1)+1$ since we have b interleaved DRAM. It *doubles* the size of tail SRAM, but the memories are distributed and we can employ static allocations for each SRAM as well.

We can see that the time complexity to handle every packet is $O(1)$ since it has the definite DRAM to go and every packet only makes its reservation once. Packet transmissions can be desynchronized. It further improves the promptness of arriving packets. This will especially benefit burst traffics.

Another practical consideration is that we assumed heavy traffic in the proof of the theorem. In practice in the light traffic conditions, the CM MMA should be modified slightly. The examination process will end whenever we find a maximum matching or reach the end of packet queue. This will also provide the CM MMA with the ability to cope with bursty traffic. The CM MMA will try to transfer packets at the silence of the arriving packets, not necessarily to wait until b packets are ready.

VII. Conclusions

To make fast and large network memories, we employ a hybrid SRAM/DRAM architecture with interleaved DRAM in the middle for main packet storage and the SRAM in the tail (head) for write (read) buffering.

We designed a deterministic memory management algorithm to provide hard performance guarantees to the memory system. In particular, we can provide more scalability of the memory system and less average delay with the same SRAM size requirement as that in previous work in [1].

We believe the CM-MMA algorithm is ready to be implemented in hardware and capable of building high-performance network memories for several generations of technologies to come.

Reference

[1] S. Iyer, R. R. Kompella, and N. McKeown, "Analysis of a memory architecture for fast packet buffers," *IEEE Workshop on High Performance Switching and Routing*, 2001.

[2] H. J. Chao, "Next generation routers," *invited paper, IEEE proceeding*, vol. 90, 2002.

[3] G. Appenzeler, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *ACM SIGCOMM*, 2004.

[4] "http://www.samsung.com."

[5] G. Shrimali and N. McKeown, "Building packet buffers using interleaved memories," *IEEE Workshop on High Performance Switching and Routing*, 2005.

Appendix

Complete Proof of Lemma 1 and Lemma 2

We prove the two lemmas together.

Consider a DRAM *j*. Assume the CM-MMA has run for n rounds and the SRAM is full with $Q(b-1)+1$ packets.

Therefore, there are $Q(b-1)+1+nb$ packets in total, which have arrived in the memory system. Considering a arbitrary DRAM *j*, we analyze how many packets could reserve the DRAM *j*.

We use q_i to denote the number of packets belonging to the *i*-th FIFO queue. Then q_i can always be represented by the following form:

$q_i = n_i b + m_i (0 \le m_i < b)$

This form tells that at least n_i packets and at most $n_i + 1$ packets from the *i*-th FIFO queue reserve the DRAM *j*, since all the packets are assigned to the DRAM in a round robin way.

Then $\sum_{i=1}^{Q} q_i = Q(b-1)+1+nb$.

That is to say:

$\sum_{i=1}^{Q} n_i b + \sum_{i=1}^{Q} m_i = Q(b-1)+1+nb$

$\Rightarrow \sum_{i=1}^{Q} n_i b = Q(b-1)+1+nb-\sum_{i=1}^{Q} m_i$

$\Rightarrow \sum_{i=1}^{Q} n_i = \frac{Q(b-1)+1+nb-\sum_{i=1}^{Q} m_i}{b}$

Since $\max(\sum_{i=1}^{Q} m_i) = Q(b-1)$, we can see that

$\min(\sum_{i=1}^{Q} n_i) \ge \frac{1+nb}{b} = n + \frac{1}{b}$

Since n_i are integers, $\min(\sum_{i=1}^{Q} n_i) \ge n+1$.

This formula indicates that among the $Q(b-1)+1+nb$ packets there are at least $n+1$ packets that reserve the DRAM *j*. Since every round matches exactly one packet to the DRAM *j*, after *n* rounds, there is at least one packet left that reserves the DRAM *j* in the $Q(b-1)b+1$ packets. This proves lemma 1.

It is obvious to see that for $q_i = n_i b + m_i (0 \le m_i < b)$ packets in the *i*-th FIFO queue, there are at most $n_i + 1$ packets which can reserve the DRAM *j*, and $m_i \ge 1$.

Thus, we can calculate

$$\max(\sum_{i=1}^{Q} n_i) \le \frac{Q(b-1)+1+nb-Q}{b} = Q\frac{b-2}{b} + \frac{1}{b} + n$$

$$\max(\sum_{i=1}^{Q} (n_i+1)) \le Q\frac{b-2}{b} + \frac{1}{b} + n + Q$$

$$= \frac{2Q(b-1)+1}{b} + n$$

Using the same analysis as above, we can see that there are at most $\frac{2Q(b-1)+1}{b}$ packets that reserve the DRAM *j* after *n* rounds. This proves lemma 2.