Optimizing Memory Bandwidth of a Multi-Channel Packet Buffer

Sarang Dharmapurikar Sailesh Kumar John Lockwood Patrick Crowley
Dept. of Computer Science and Engg.
Washington University in St. Louis, USA.
{sarang,sailesh,lockwood,pcrowley}@arl.wustl.edu

***Abstract*— Backbone routers typically require large buffers to hold packets during congestion. A thumb rule is to provide a buffer at every link, equal to the product of the round trip time and the link capacity. This translates into Gigabytes of buffers operating at line rate at every link. Such a size and rate necessitates the use of SDRAM with bandwidth of, for example, 80 Gbps for link speed of 40 Gbps. With speedup in the switch fabrics used in most routers, the bandwidth requirement of the buffer increases further. While multiple SDRAM devices can be used in parallel to achieve high bandwidth and storage capacity, a wide logical data bus composed of these devices results in suboptimal performance for arbitrarily sized packets. An alternative is to divide the wide logical data bus into multiple logical channels and store packets into them independently. However, in such an organization, the cumulative pin count grows due to additional address buses which might offset the performance gained. We find that due to several existing memory technologies and their characteristics and with Internet traffic composed of particular sized packets, a judiciously architected data channel can greatly enhance the performance per pin. In this paper, we derive an expression for the effective memory bandwidth of a parallel channel packet buffer and show how it can be optimized for a given number of I/O pins available for interfacing to memory. We believe that our model can greatly aid packet buffer designers to achieve the best performance.**

I. Introduction

Backbone routers provide packet buffers which can hold all the data transmitted in a round trip time. With a round trip time of 0.25s, this translates into a buffer of $40Gbps \times 0.25s = 10Gbtes$ as observed in [3]. Further, this buffer must be able to operate at twice the line-speed i.e. at 80Gbps data rate. Typically such buffers are also coupled with per-flow queuing system in which millions of packet queues are maintained. Packets are queued in the associated queue as they arrive and a scheduler reads packets from queues according to some algorithm such as Deficit Round Robin or Weighted Round Robin. The most commonly used approach is to buffer the packet data and use the tag assigned to the packet for manipulations such as queuing and dequeuing. Such packet buffers are hard to engineer with high-speed low-density SRAM since it is not only costly but also impractical due to very high power consumption. Therefor, high-density and low-power SDRAM is used for this purpose. Unfortunately SDRAM operates slower than SRAM, in terms of speed at which it can read or write data as well as the latency incurred in each memory operation.

Designing an SDRAM based packet buffer which is *scalable* to high link-speeds is challenging for several reasons. The question we seek to answer in this paper is: How can we use multiple DRAM memory modules in parallel to achieve an arbitrarily large throughput? It might appear at first that the slow speed of SDRAM can be compensated for by using multiple SDRAM modules in parallel to make a wider logical data bus to achieve the desired throughput. However this is not true in practice. This can be explained with a simple example. Assume that we can write to SDRAM at full speed without incurring any latency, i.e., we exploit the full raw bandwidth of the SDRAM. To achieve a raw bandwidth of 12.8 Gbps, a SDRAM memory module with 128 bit wide data bus which can read/write data at 100 MHz can be used. If the length of the packet to be written is 128 bits then in a single clock cycle a 16 byte long word can be read or written in the SDRAM. With this granularity of access it takes 5 clock cycles to do this operation. However, if the packet length is 81 bytes then it takes 6 clock cycles. In the sixth clock cycle only a single byte is written, with the other 15 bytes in the same word being unused. We can not insert the data from the next packet in the unused portion of the word because the next packet might not belong to the same queue that the current packet belongs to. On the other hand, we can fill the unused word with the the initial bytes from the next packet destined for the same queue although we must wait until that next packet arrives. In this case, the writing of the word must be delayed until the word is completely filled and the remaining data must be kept in buffer memory in the interim. Moreover, this buffering must be done on a per-queue basis. This wait time increases with bus width. However as soon as enough data on that queue is received in the buffer, it can be written in the SDRAM. This is precisely the model used by some of the existing packet buffer architectures including [3], [2]. To summarize this scheme, a packet buffer is maintained in an on-chip SRAM memory for each queue and as soon as enough packets are accumulated in any queue, the batch of packet is written to the SDRAM queue. Similarly, on the read side, a batch of packets is read from the SDRAM queue and buffered in an on-chip SRAM from which they depart, one or more in a scheduling interval.

While this scheme eliminates the original problem of data-bus under-utilization, it introduces another critical problem. The on-chip SRAM buffering requirement grows linearly with the number of queues, rendering it impractical for large num-

ber of queues. Indeed, the authors of [3], [2] acknowledge this problem and call for an alternative architecture for supporting a large number of queues.

In this paper we explore an alternative approach to building a high throughput packet buffer, namely using multiple SDRAM modules. Our basic underlying idea is to split the buffer data bus into multiple smaller channels which can be accessed in parallel. Each incoming packet can be stored in one of the channels and thus multiple packets can be stored in multiple channels concurrently. The segmentation of the data bus in this fashion not only improves utilization but also obviates the need for a large on-chip SRAM cache. We highlight a few design issues in engineering this type of buffer and provide solutions to some of them. The issue that is of specific interest to us in this paper is: Given a fixed number of I/O pins on a chip to interface with multiple SDRAM modules, how many parallel data channels should be created with them such that the bandwidth per pin is maximized?

The paper is organized as follows. In the next section we discuss the alternative model of a packet buffer based on parallel channels. In Section III we provide the quantitative analysis of the engineering trade-offs. In section IV we evaluate these trade-offs by considering realistic values of parameters involved. Section V concludes the paper.

II. Parallel Channel Buffer

As we observed earlier, poor utilization of a wide memory channel arises because the last word of the packet can be partially filled. Recalling our earlier example, on a 128 bit data bus, writing an 81 byte packet requires 6 clock cycles with a wastage of 15 bytes in the last word. This wastage of bandwidth would have been avoided if we could have accessed the memory at a finer granularity by splitting the bus into 16 different channels each a single byte wide. In this arrangement, a single 81 byte packet can be written to one channel in 81 clock cycles. 16 such packets can be written into 16 different channels, simultaneously, in 81 clock cycles. This would provide 100% efficiency.

However, by doing this we introduce another overhead: the overhead of independent address pins required to access each parallel channel. The more independent channels we create out of a single homogeneous channel, the more I/O pins will be required for independent address buses. If we had a fixed budget of I/O pins then these pins must be used carefully to give a high memory bandwidth. Having more address pins consumes more I/O pins of the chip, leaving fewer I/O pins for data input-output and thereby reducing memory bandwidth. Therefore, between the two extremes of having a single wide memory with just one address bus (but under utilization of the channel) and having many parallel channels (but wasting pins in address buses), there lies an optimal point where the number of channels is just right such that the under utilization and address pin overhead are both minimized for a given packet size distribution. In the next section, we quantify these trade-offs to make an efficient choice of memory configuration.

III. Trade-offs in Memory Configuration

We begin with a set of memory configuration parameter definitions.

- p : the total number of pins available for the I/O
- n : the number of parallel channels the data bus is partitioned into
- w : the channel width in bytes
- a : address pins(including other overhead such as command bus) for a single memory channel
- f_{clk} : the frequency (in MHz) at which memory operates
- T_{clk} : $1/f_{clk}$, the time period of a clock (in ns)

It should be noted that Dual Data Rate (DDR) SDRAM can clock data at both positive and negative clock edges and hence gives twice the data rate of Single Data Rate (SDR) SDRAM. Today DDR-SDRAM is the most popular option and our analysis will be based on this technology. The frequency f_{clk} represents the original frequency of DDR-SDRAM (which is typically 200 MHz for modern DDR-SDRAM chips) which means in every T_{clk} time we can have two words read or written in the memory.

Using the notation above, we can express the width of the channel as

$$w = \left\lfloor \frac{p - na}{8n} \right\rfloor \tag{1}$$

In the equation above, na indicates the address bus pins used for n channels each of which needs a address pins. After subtracting these pins from the total available pins the leftover pins are partitioned between n channels. The factor of eight in the denominator is used to simply represent the channel width in bytes.

The terms used above are illustrated in Figure 1. As the figure shows, due to rounding off, some pins are left unused. Note that not all parameter choices are rational, since no practical memory organization would leave pins unused (these would simply be discarded).

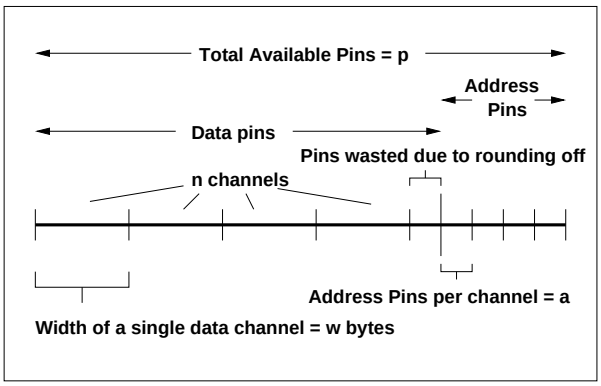

Fig. 1. Distribution of the available I/O pins

The raw memory bandwidth of the system is maximum when the data path is a single wide channel having a single address bus. In this case, the minimum number of pins are used for the address bus and the rest of the I/O pins are used for data bus.

However, as discussed in the earlier section, wider channels also yield a poor efficiency.

A. Modeling Memory Bandwidth

We will now develop an expression for the efficiency of the data bus for a given number of pins and a given channel width configuration. First we express the efficiency of a single memory channel with the parameters defined. To do so, we explain the details of how a single memory channel operates. It should be recalled that for all practical purposes, a single channel is equivalent to a single homogeneous SDRAM memory with wider data bus and bigger size.

To accurately model SDRAM, we must discuss an additional source of inefficiency. Memory elements in SDRAM are arranged in 3-dimensions, namely rows, columns and banks. When a memory operation accesses a row in a bank, the next memory operation on a different row of the same bank can not be executed for a minimum time period commonly known as t_{RC}. We refer to the event of two memory operations of the same type on the same bank and different row as a *bank collision*. Further, if the collision happens within the latency of t_{RC} then we refer to this collision as a *conflict*. Since bank conflicts introduce some latency in back-to-back memory accesses, they can significantly degrade the memory bandwidth.

In the context of packet buffering, we note that the process of writing a packet is under our control and we are free to choose the bank to write to; that is, when writing a packet buffer, the memory controller can choose destination addresses, and therefore banks, so as to avoid conflicts. However, the reading of packets is dictated by a scheduler which can request packets from any bank according to its policy. In order to avoid bank conflicts, we interleave the read and write memory accesses and always force the write memory access to a different bank than the previous read access and the following read access. This is illustrated in the figure 2.

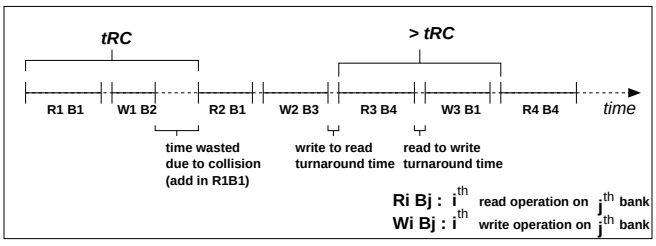

Fig. 2. Interleaving read and write operations on a memory channel

As the figure shows, the write operations are always scheduled on a different bank than those used by the previous and following read operations. By inserting a write operation (on a bank of our choice) between two reads we not only avoid two consecutive operations on the same bank but also attempt to make sure two successive reads do not conflict (e.g., R3B4-W3B1-R4B4 in Figure 2). However, the write operation duration might not always be sufficient to maintain the t_{RC} separation. This is particularly the case with small packets when written on a wide channel (which hence require a very short time). In such cases the conflicts can not be avoided (e.g., R1B1-W1B2-R2B1) and the following read access must wait until the t_{RC} elapses.

In order to express the efficiency of a single memory channel, we use a notion of *work*. Work is measured in bytes and represents the amount of byte-level pin activity carried out to read or write a packet. An l byte packet ideally requires a minimum of l units of work. However, due to the inefficiencies introduced by odd channel widths, burst lengths, turnaround latencies and t_{RC}, excess work beyond the minimum is performed. We now explain each source of inefficiency in greater detail.

Odd channel width: Suppose a packet of size l bytes is to be written into, or read from, a single channel. Since the packet must be written in the form of multiple words, each word being the size of a channel width (w), we use $\lceil l/w \rceil$ words with the last word possibly having some unused bytes. This is illustrated in Figure 3 in which we wish to write five words in a four byte channel. Thus the second word has three unused bytes leading to some inefficiency.

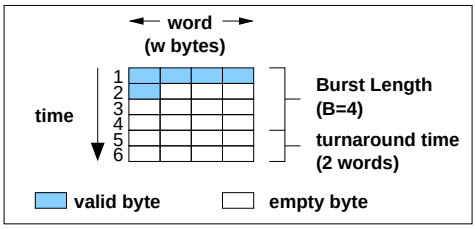

Fig. 3. Illustration of ineffi ciencies arising due to channel width, burst length and turnaround latency. To write a 5 byte data we spend time worth 24 bytes.

Burst Length: Since accessing individual random words in an SDRAM is inefficient due to t_{RC} delays, all SDRAM transactions are typically performed in bursts where a set of words from the same row of a bank is accessed back-to-back. The burst length is often chosen to be 2 words, 4 words or 8 words and is frequently configurable. Any transaction requiring a burst length other than the given choices must be expressed as a combination of them. Thus, a burst length of 6 can be achieved by a burst length of 4 and 2 together. It should be noted that in DDR-SDRAM memories the smallest burst length must be 2 since in a single clock cycle at least 2 words can be read or written. We will assume a burst length of B for executing our transactions on each channel. In this burst, the last few words can be empty leading to some loss of memory bandwidth. In the example of Figure 3, in which B=4, we must complete a 4-word operation which results in the third and fourth words being empty.

Turnaround latency: As illustrated in Figure 2, the channel data bus is idle for some period due to turnaround latency. This latency is typically a clock cycle (equivalent to 2 words) for modern DDR-SDRAMs [4]. Thus, each transaction involving l bytes requires two additional words worth period as shown in Figure 3.

We can now explain the concept of work with Figure 3. In order to write 5 bytes of data, the total work done was $4 \times (4 + 2) = 24$ bytes. Here the number in parentheses indicates the number of *word-transactions* involved. We define *efficiency* to be the ratio of work needed to work actually done. Generalizing this, with a burst length B and turnaround latency of 2 words, the total number of word-transactions of a l byte packet can be expressed as

$$t(l, B) = \left\lceil \frac{\lceil l/w \rceil}{B} \right\rceil B + 2 \tag{2}$$

From this point onwards, we develop separate expressions for word-transactions during read and write operations. It should be recalled that the write operation can be directed to any bank of our choice. Hence, we can ensure that two successive write transactions go to different banks and effectively hide the t_{RC} delay. Therefore, the number of word-transactions needed for an l byte write operation is the same as is expressed in the equation above. Each word-transaction is equivalent to w bytes of work. Hence the work performed in write operation is

$$W_{write}(l, B, w) = t(l, B)w \tag{3}$$

Now, consider a sequence of N packets each of size l_i bytes, which is to be written into a single channel. The work needed in this case is $\sum_i l_i$. Actual work performed is $\sum_i W_{write}(l_i, B, w)$. Hence, efficiency can be expressed as

$$\begin{aligned} \rho_{write}(B, w) &= \frac{\sum_{i=1}^{N} l_i}{\sum_{i=1}^{N} W_{write}(l_i, B, w)} \\ &= \frac{\sum_{i=1}^{N} l_i/N}{\sum_{i=1}^{N} W_{write}(l_i, B, w)/N} \\ &= \frac{\sum_l (N_l/N) l}{\sum_l W_{write}(l, B, w) N_l/N} \end{aligned} \tag{4}$$

where, N_l indicates the number of packets of length l among N packets. As $N \to \infty$, the ratio $N_l/N = P(l)$ where $P(l)$ is the probability of the occurrence of an l byte packet in the data stream. Hence, we can write the efficiency over a large number of packets as follows.

$$\rho_{write}(B, w, P) = \frac{\sum_{l=l_{min}}^{l_{max}} lP(l)}{\sum_{l=l_{min}}^{l_{max}} W_{work}(l, B, w)P(l)} \tag{5}$$

where l_{min} and l_{max} are the minimum and maximum packet lengths, respectively. The inclusion of parameter P in ρ indicates its dependence on the packet length distribution. By accounting for the probability of occurrence of each packet size, the above expression naturally quantifies the efficiency of a memory channel configuration for a given packet distribution.

We now express the efficiency for read operations. The primary difference between the read and write operations is that the order and location of reads are dictated by the scheduler and can cause collisions (e.g., R1B1-R2B1 and R3B4-R4B4 in Figure 2). In the case of a bank collision, if there is no conflict (e.g., R3B4-W3B1-R4B4), then the read operation effectively consumes the same word-transactions as $t(l, B)$. In the case of a conflict, the second read operation must wait until t_{RC} time elapses (e.g., R1B1-W1B2-R2B1). We add this wasted time to the time needed by the first read operation (R1B1 in the example). The number of word-transactions possible in time t_{RC} are

$$k = 2 \times t_{RC}/T_{clk} \tag{6}$$

The factor of 2 in the equation above appears due to Dual Data Rate assumption. To calculate the effective word-transactions of a *conflicting* read operation (e.g., R1B1), we must subtract the word-transactions corresponding to the following write operation (W1B2) from k. However, word-transactions for this write depend on the length of the packet being written. We will take a conservative bound of minimum packet length (l_{min}) for this write operation. Hence the effective number of word-transactions for a *colliding* read operation is essentially the maximum value among the word-transactions for colliding but not conflicting operation and colliding and conflicting operation. This can be expressed as

$$t^c_{read}(l, B) = max\left\{(k - t(l_{min}, B)), t(l, B)\right\} \tag{7}$$

The superscript c in the equation above indicates a collision. Without any collisions among read requests (R2B1-W2B3-R3B4), the word-transactions per read operation are the same as $t(l, B)$. Now, assuming that the read bank access pattern is sufficiently random, we encounter a collision with probability $1/b$ where b is the number of banks. With these considerations, the work done on average for each read operation for l-byte packets can be expressed as

$$W_{read}(l, B, w, b) = \left(t^c_{read}(l, B)\frac{1}{b} + t(l, B)\frac{b-1}{b}\right) w \tag{8}$$

giving the efficiency

$$\rho_{read}(B, w, b, P) = \frac{\sum_{l=l_{min}}^{l_{max}} lP(l)}{\sum_{l=l_{min}}^{l_{max}} W_{read}(l, B, w, b)P(l)} \tag{9}$$

The overall efficiency of a single channel is

$$\rho(B, w, b, P) = \frac{\rho_{write}(B, w, P) + \rho_{read}(B, w, b, P)}{2} \tag{10}$$

Now, we can express the average memory bandwidth of the system $r(B, p, n, b, P)$ as

$$r(B, p, n, a, b, P) = 8wn\rho(B, w, b, P)f_{clk} \qquad Mb/s \tag{11}$$

Note that we have replaced w in the left hand size by p, n and a since it can be expressed as their function using Eq. 1. To summarize, the throughput of a parallel channel packet buffer depends on the number of pins p, the number of channels n, the address pins used for each channel a, the number of banks in each SDRAM channel b, the burst length of operation B and finally the packet length distribution P.

IV. Evaluation

In this section we consider some realistic values of the parameters involved in Eq. 11 to evaluate the memory bandwidth. We will consider burst length $B = 4$ since burst length shorter than this works against us in meeting the t_{RC} delay. The number of banks are minimum $b = 8$ in most of the modern DDR-SDRAM. The number of address pins are typically below 20 since DDR-SDRAM use address multiplexing on

a narrow address bus to clock row and column addresses. Considering the overhead of the command bus (typically 3 pins), and other overheads such as chip select etc. we fix the parameter $a = 20$. Finally, we need to consider a particular packet length distribution too for evaluation. We consider the packet length distribution of the Internet packets which we obtained form the traces available at CAIDA [1]. Typical Internet packet length distribution is shown in Figure 4. We refer to this distribution as P_{avg} since it reflects the average traffic. As the figure shows, Internet traffic is dominated by 40 byte packets (ACK packets of TCP) and 1460 byte packets (typically the maximum transfer unit in the network).

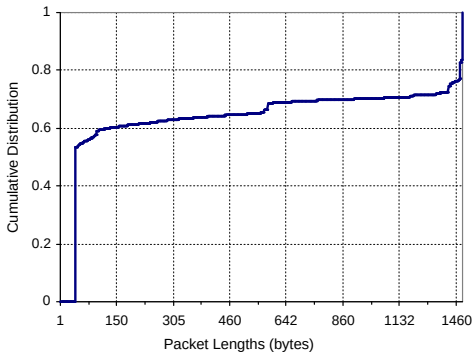

Fig. 4. Typical packet length distribution of Internet packets

With these parameters, we plot the bandwidth of the packet buffer versus the number of channels for different number of pin budget as shown in Figure 5. In other words, we plot $r(B = 4, p, n, a = 20, b = 8, P = P_{avg})$ with n on x-axis and each curve corresponding to a particular value of p. To simplify the expression we drop $B = 4$, $b = 8$, $a = 20$ and represent it as $r(p, n, P)$.

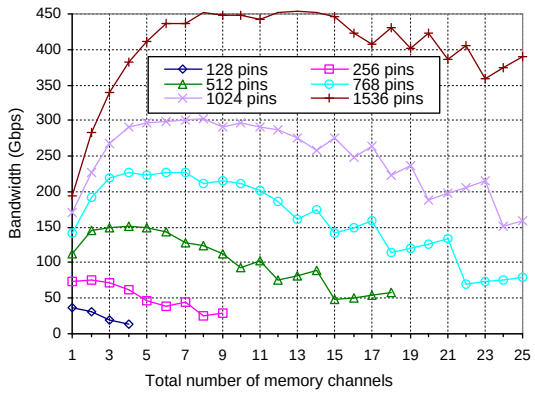

Fig. 5. Average buffer bandwidth as a function of the number of channels for different pin budgets.

It is evident from this plot that the bandwidth increases initially as the number of channels is increased since under-utilization of wide data bus is reduced with narrower channels. Then the maxima is reached at certain number of channels followed by eventual decline in bandwidth when the effect of pin wastage in address buses dominates. Clearly, the maxima indicates the best choice of the number of channels for the given pin budget. In Figure 6, we plot the maxima of each curve in Figure 5, which indicates the peak bandwidth achieved with varying pin budgets. In other words, we plot $Max_n\{r(p, n, P = P_{avg})\}$ against p. As the figure shows, the maximal bandwidth increases linearly with the optimal choices of configurations for each pin budget. Moreover, this figure also also shows at how many channels $n = n_{opt}^{avg}$, this optimal point is achieved for the average traffic pattern. Thus, with 576 total pins, we must create $n = 4$ channels to get the best bandwidth of over $160Gbps$ for the average case traffic pattern and hence $n_{opt}^{avg} = 4$.

It should be noted that the configuration of n_{opt}^{avg} channels for p pins will maximize the bandwidth under the typical Internet traffic(i.e. $P = P_{avg}$) and not under the worst-case traffic. Now, an interesting question that arises is: after designing a packet buffer to maximize the bandwidth under average traffic, how would it perform in the face of worst case traffic pattern? To answer this question, we must define what worst case traffic looks like. Channel performance is worst when packet length is just one byte more than the smallest multiple of the channel width (and also > 40 bytes). Given a channel width we can always design a packet which meets this criterion and bombard the buffer with these packets to degrade its efficiency. This type of traffic with a unique packet length is what we call the worst case traffic. Clearly, this worst case traffic differs for each channel width configuration. With this consideration, the third curve in Figure 6 plots the worst case bandwidth for a configuration optimized for P_{avg} i.e., it plots $Min_P\{r(p, n = n_{opt}^{avg}, P)\}$. For instance, when we design our buffer with 576 pins to maximize bandwidth for average traffic pattern there exists a packet length by which we can reduce the bandwidth of this system to about $80Gbps$, provided traffic consists of packets only with this length. Though this gives an idea of a worst case bandwidth, readers should note that such a traffic scenario is very hard to contrive for any adversary.

It is clear that when the packet buffer is optimized for the average case performance, a small number of channels are needed, which in turn results in poor worst-case performance.

So far we considered a configuration to maximize bandwidth for average traffic and also evaluated its performance when faced with worst case traffic. Now we first design the buffer to handle the worst case traffic and then evaluate its performance when used for average traffic. As we mentioned before, a channel shows a worst case performance when we read/write packets of length which is just one byte more than the smallest multiple of channel width (and also > 40 bytes). First we plot the worst case bandwidth for a given number of pins and given number of channels by contriving the worst case traffic for that channel width. Thus, we essentially plot $Min_P\{r(p, n, P)\}$. This is shown in Figure 7. We note that the maxima is generally achieved for far greater number of channels compared to the average case.

Then in Figure 8, we plot each maxima point that was

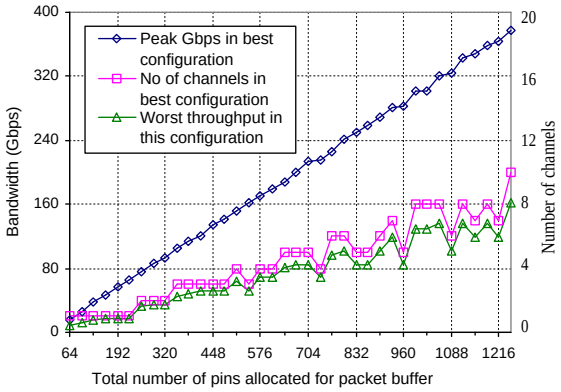

Fig. 6. Memory bandwidth as a function of pins when the buffer is designed to maximize the average performance

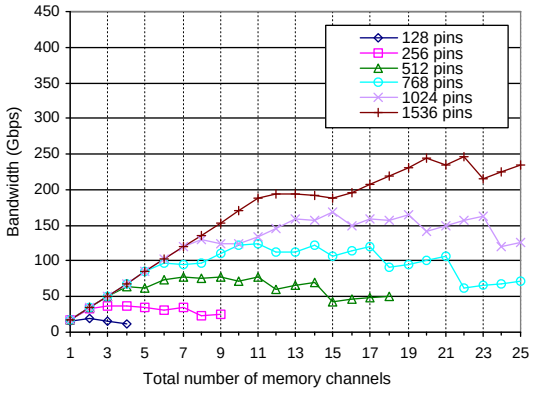

Fig. 7. Worst-case memory bandwidth as a function of number of channels for different pin budgets

observed in Figure 7 for different pin budgets, i.e., we plot $Max_n\{Min_P\{r(p,n,P)\}\}$. For an example, the figure shows that with 1024 pins, we can design a buffer for which the worst case bandwidth will can never be less than 170 Gbps no matter what the worst case traffic is. Again we also plot the number of channels needed to achieve this maximal worst-case performance $n = n_{opt}^{worst}$. From the figure, $n_{opt}^{worst} = 15$ for 1024 pins. After fixing the optimal number of channels for the maximal in worts case, we evaluate its performance for average case traffic pattern ($P = P_{avg}$). Formally, we plot $r(p, n = n_{opt}^{worst}, P = P_{avg})$. With our previous example, if we had 1024 pins then we would create 15 channels for optimizing the worst case and ensure that the bandwidth is never less than 170 Gbps. The same configuration will give a bandwidth of around 320 Gbps when faced with the average case traffic.

We believe our plots are comprehensive and exhibit a methodology to model and optimize the memory bandwidth of packet buffers.

V. Conclusions

In this paper we modeled the bandwidth efficiency of a multi-channel packet buffer in which multiple parallel memory

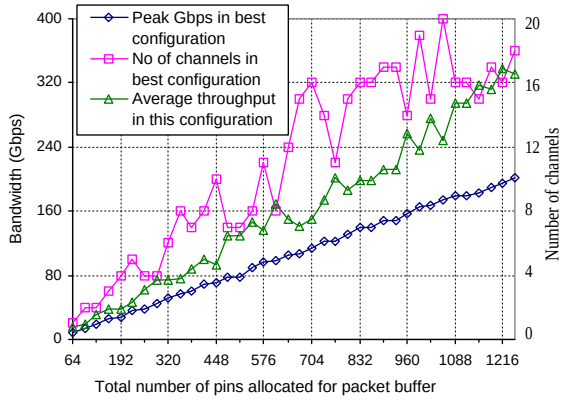

Fig. 8. Memory bandwidth as a function of pins when the buffer is designed to maximize worst case performance.

channels can be accessed concurrently. We show that this bandwidth critically depends on the number of channels we create using a fixed budget of I/O pins to interface the external memory. It also depends on the memory technology used and packet length distribution. Our model not only captures the impact of these different parameters on bandwidth but also shows how to maximize it for a given packet length distribution by choosing the number of channels judiciously. We have used this model with the realistic values of memory technology related parameters and evaluated the bandwidth for average packet length distribution as well as worst case packet length distribution. We believe that our model and methodology can be a powerful aid for the system designers to make crucial design decisions.

VI. Acknowledgments

This work was supported in part by NSF grants CCF-0430012 and CNS-0325298 for which we are grateful.

References

[1] http://www.caida.org.
[2] I. K. G. Shrimali and N. Mckeown. Designing Packet Buffers with Statistical Guarantees. In *Hot Interconnects-12*, August 2004.
[3] S. Iyer, R. R. Kompella, and N. McKeown. Designing buffers for router line cards. Technical report, Stanford University HPNG Technical Report, 2002.
[4] Micron Inc. Double data rate (DDR) SDRAM MT8VDDT6464HD 512MB data sheet, 2004.