CS–1997–20

Balancing DMA Lantency and Bandwidth in a High-Speed Network Adapter

Kenneth G. Yocum Darrell C. Anderson
Jeffrey S. Chase Syam Gadde
Andrew J. Gallatin Alvin R. Lebeck

Department of Computer Science

Duke University

Durham, North Carolina 27708–0129

November 1997

Balancing DMA Lantency and Bandwidth in a High-Speed Network Adapter

Kenneth G. Yocum Darrell C. Anderson Jeffrey S. Chase Syam Gadde
Andrew J. Gallatin Alvin R. Lebeck

November 1997

Abstract

Advances in networks and I/O bus structures continue to improve the latency and bandwidth of communication in computer clusters. While high-end PCI I/O systems have improved I/O performance significantly in recent years, I/O bus transfers must be managed carefully to deliver the potential of Myrinet and other gigabit networks.

This paper defines and evaluates design choices for DMA management in high-speed network adapters. In particular, we descibe a combination of buffering and DMA policies referred to collectively as *adaptive cut-through delivery*, and explore its behavior on a wormhole-routed Myrinet network using four workstation platforms with varying host-PCI bridge implementations. We show that adaptive cut-through delivery combines low latency for large packets (e.g., 4K transfers take 70μs on 300 MHz Pentium-II/440LX platforms) with high bandwidth under load (up to 126 MB/s). Our experiments use Trapeze, a messaging system designed for network memory and other cluster operating system services.

1 Introduction

Network performance is a critical factor in the effectiveness of cluster environments for parallel computing, scalable network services, distributed file storage, and network memory. These systems need a messaging substrate that delivers the performance of the network and I/O system hardware. The messaging layer must support communication styles appropriate to the application, with a mix of large and small messages. It must handle network traffic with minimal overhead, balancing low latency with high bandwidth.

Since many parallel programs generate large numbers of small messages [19], much recent research has sought to minimize messaging latency and overhead by bypassing the operating system and network protocol processing through memory-mapped network interfaces (e.g., Hamlyn [6], U-net [2], SHRIMP [3], Active Messages [9], FM [22]). To a large degree these efforts have been successful, and the focus of research has now shifted to supporting multiple applications safely over these low-overhead interfaces [25, 13, 8].

This paper addresses a complementary area that has received less attention: managing data movement between the host and the network fabric through the network interface. We explore data transfer choices for network interfaces connected to the I/O bus; while this is a lower-cost solution than a direct connection to the memory bus, it demands careful handling of I/O bus transfers to minimize the potential of the I/O bus as a bottleneck. Several studies have addressed these issues [11, 12, 24]. We report on experiences with these issues in the context of newer I/O and networking technologies: the gigabit PCI I/O bus standard [16], and Myricom's 1.28 Gb/s Myrinet [4] network.

Our work uses the Trapeze messaging interface [21] on a Myrinet cluster of Intel Pentium platforms and Digital Equipment Corporation AlphaStations using PCI I/O buses. PCI has significantly expanded the potential for high-speed networking on workstation-class systems. Our experiments use 32-bit PCI buses clocked at 33MHz, delivering peak bandwidths of 120 MB/s to 132 MB/s; bandwidth will continue to increase with the advent of 64-bit PCI implementations clocked at 66MHz or higher. As one of the first groups to use Myrinet on high-end PCI platforms, we have been able to run Trapeze at point-to-point bandwidths up to 126 MB/s.[1] This requires a highly streamlined firmware implementation for the Myrinet network interface cards (NICs), with careful handling of data transfers to and from the NICs.

This paper makes three contributions. First, we explore and illustrate the effects of several alternatives for scheduling DMA transfers on high-speed network interfaces. Second, since the behavior of these schemes is highly dependent on the characteristics of the host I/O system, and particularly the host-PCI bridge, we report experiences on platforms based on the DEC Pyxis, DEC CIA, and Intel 440LX and Intel Natoma chipsets. These systems exhibit a variety of PCI bridge characteristics, giving insight into the performance of workstation I/O systems and their effect on network communication.

Most importantly, we explore the behavior of *adaptive cut-through delivery*, a combination of techniques for buffering and data transfer present in some network switches and adapters. Adaptive cut-through delivery combines the highest achievable bandwidth with low latency for messages larger than a few hundred bytes; it is related to cell batching and pipelining techniques in some high-speed ATM adapters [10]. Used with Myrinet, adaptive cut-through delivery pushes 4K transfer times below 80μson high-quality PCI platforms. This has potential to significantly reduce I/O stall times for I/O intensive applications running in clusters, an environment in which low latency for large packets is particularly important. We believe that adaptive cut-through delivery is a key technique in high-speed network interface design, that this technique has not previously been explored in the literature, and that our use of adaptive cut-through delivery is unique among published firmware implementations for Myrinet, the most widely used network for research on fast messaging today.

The paper is organized as follows. Section 2 gives an overview of the Trapeze messaging system and the Myrinet hardware and PCI platforms we use. Section 3 explores the behavior of DMA scheduling in the adapter, with particular focus on adaptive cut-through delivery, and presents the results of large-packet latency and bandwidth benchmarks using Trapeze. We conclude in Section 4.

2 Trapeze Messaging for Myrinet

Our Myrinet configuration uses PCI-connected Myrinet network interface cards (NICs), and full-crossbar network switches. The Myrinet NICs (M2F-PCI32) include a block of dual-ported static RAM and a custom CPU and link interface based on the LANai 4.1 chipset, clocked to the 33 MHz PCI bus. The Myrinet NIC is controlled by a firmware program written into the adapter SRAM by the host at startup. The switches are derived from wormhole-routed multiprocessor interconnect

[1]Measured user-to-user for streams of 64K packets on pairs of Pentium-II/440LX systems. At the moment this is the highest bandwidth yet reported with Myrinet.

designs [23]; latency across a single switch is about 1 μs. The Myrinet links are full-duplex channel pairs delivering 160 MB/s of point-point bandwidth in each direction. We use both LAN and SAN cabling in our cluster, with no difference in point-to-point performance.

Packet data is always sent and received from buffers in the adapter memory. The adapter can act as a PCI bus master to move data between its buffers and host memory or other PCI devices using DMA. However, the entire NIC SRAM is directly addressable in the host physical address space using programmed I/O (PIO). Since message buffers on the adapter are also addressable, the host may read or write message contents using PIO rather than DMA. Trapeze uses a mix of PIO (see Section 2.3) and DMA; this paper deals primarily with the techniques for handling DMA transfers.

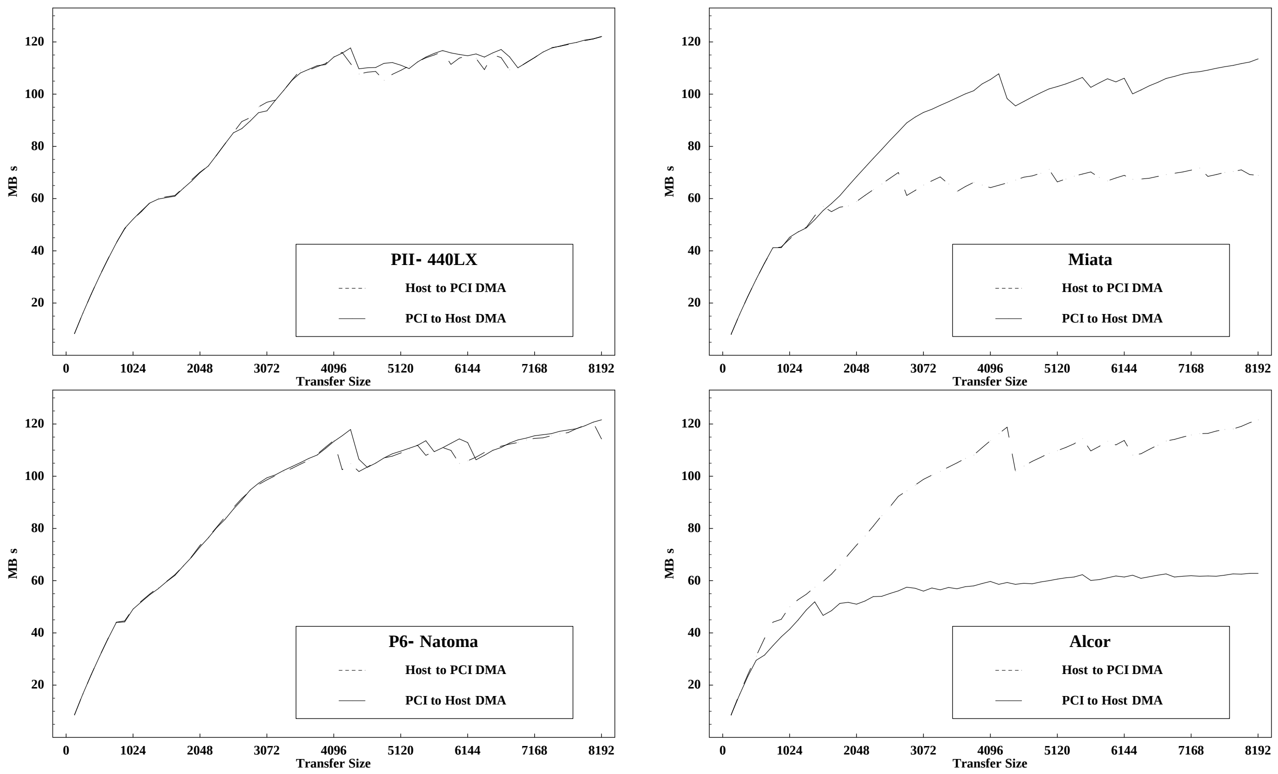

Figure 1: DMA performance of four workstation platforms.

Communication performance with Myrinet is primarily limited by PCI implementations on the market today, and the handling of transfers on the PCI bus has a significant effect on communication performance for Myrinet and other high-speed networks, today and for the forseeable future. To illustrate the issues, we report on experiences using Myrinet with four different Intel and Digital Alpha platforms:

- Digital AlphaStation 500 and 600 ("Alcor") running Digital Unix 4.0. Alcor has a 266 MHz 21164 CPU and a host-PCI bridge known as CIA (ASIC pass 2).
- Digital AlphaStation PWS 500au ("Miata") running Digital Unix 4.0. The CPU is a 500 MHz Alpha 21164 CPU with a Pyxis host-PCI bridge (ASIC pass 257).

- Asus P/I-XP6NP5 motherboard with a 200 MHz Intel P6 CPU and an Intel Natoma PCI chipset, running FreeBSD 2.2.
- Dell Dimension XPS D300 workstation with a 300 MHz Pentium-II CPU and an Intel 440LX chipset, running FreeBSD 2.2.

Figure 1 shows the PCI bandwidth measured for DMA transfers of various sizes in the sending (DMA read from memory) and receiving (DMA write to memory) directions. The bandwidth profiles show that P6/Natoma and Pentium-II/440LX systems in pairs can deliver over 120 MB/s in both the sending and receiving directions, very close to the full bandwidth of the 32-bit 33 MHz PCI standard. For bandwidth experiments with the AlphaStations, we use an Alcor sender and a Miata receiver to circumvent bottlenecks in the current revisions of the Pyxis and CIA bridges.

2.1 Trapeze Firmware

Trapeze messages are short *control messages* with optional attached data *payloads* containing a file block, virtual memory page, or clump of application data. Trapeze payloads are sent and received using DMA to/from aligned buffers residing anywhere in host memory. The Trapeze firmware seeks to optimize handling of these payloads for low overhead in the hosts, low latency, and high bandwidth under load. The maximum payload size of a Trapeze network is configurable, and is typically set to the virtual memory page size (4K or 8K).

The Trapeze firmware and the host interact by exchanging commands and data through a shared *endpoint* structure in NIC memory. A Trapeze endpoint includes two message rings, one for sending and one for receiving, and an *incoming payload table* (IPT) that allows early demultiplexing of incoming packets on the NIC. The message rings are managed as a producer/consumer queue of short, fixed-size (128-byte) message buffers in the conventional fashion. The host uses the IPT to designate specific regions of memory as the receive buffer space for payloads attached to messages with matching tags. The IPT can be viewed as a simple, restricted form of Hamlyn's sender-based memory management [6] or the VMMC-2 incoming page table [13]. Received payloads not mapped through the IPT are deposited into a payload buffer attached to the next free entry in the receive ring.

The Trapeze firmware attempts to keep all DMA engines busy whenever there is data available to transfer. It continuously monitors status of the message rings and pending DMA activity, and initiates new transfers according to its policies for DMA handling. The LANai NIC includes three DMA engines that can manage concurrent transfers on and off the adapter: network link transmit (*netTx*), network link receive (*netRcv*), and the external PCI bus (*hostTx* and *hostRcv*). The firmware initiates DMA transfers on each engine by storing to LANai registers and then waiting for the DMA engine to signal completion by setting status bits. In addition to the status bits, the LANai exposes the progress of transfers through counter registers for each DMA engine.

2.2 Trapeze Host Interface

Trapeze was designed for direct use by the operating system kernel for kernel-based TCP/IP networking and for an RPC-like kernel-to-kernel messaging layer developed for the GMS network memory system [14]. Figure 2 depicts this structure. For benchmarking or parallel applications, the kernel may instead map the NIC SRAM into a single user process. In either mode, the host operates on

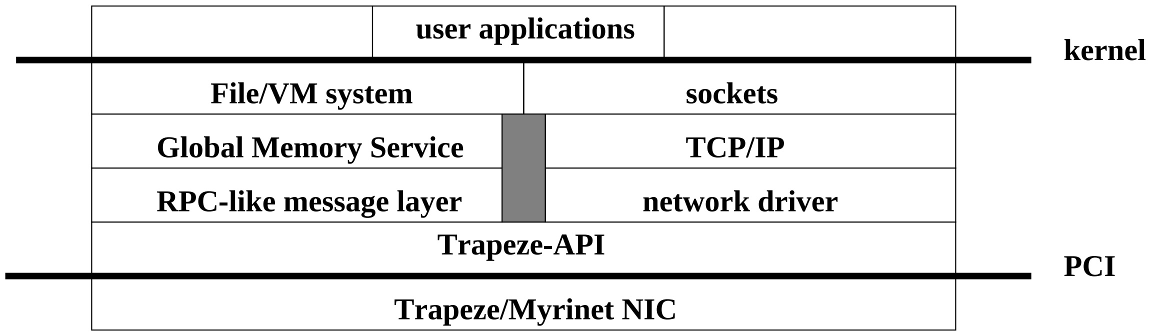

Figure 2: Using Trapeze for TCP/IP and for kernel-kernel messaging for network memory.

the network endpoint using macros and library procedures linked into the kernel or application program, and defining the lowest level API for network communication [7]. Our focus on kernel-based communication allows us to sidestep many issues that arise when accessing the network interface directly from multiple user processes, e.g., address translation, protection, virtual endpoints, receiver scheduling, notification, and page pinning [25, 13, 8].

When used within the kernel, the network driver and the RPC system share a common pool of receive buffers allocated from the virtual memory page pool, which allows unified buffering among the network, file, and VM systems. For example, GMS uses the IPT for zero-copy handling of RPC responses containing requested pages or file blocks [1]. We have also modified the Digital Unix socket layer to map received payloads directly into the receiving process when size and alignment properties allow [10, 5, 17]. For volume data transfers, the Digital Unix TCP stack passes most stream data to the Trapeze driver as aligned, pagesize buffers that can be sent as a payload to an aligned frame on the receiver. With payload remapping, this system sustains point-to-point bandwidth of 86 MB/s between AlphaStations running the standard *netperf* TCP benchmark.[2]

2.3 Latency and Overhead of Programmed I/O

The version of Trapeze measured in this paper uses PIO to move control message contents to and from the host. The intent is that the host loads and stores the message contents directly, overlapping the transfers with protocol processing, often with no intervening trip to host memory. PIO delivers lower latency at the cost of lower bandwidth and higher CPU overhead if the processor stalls.

The tradeoffs of PIO are highly platform-dependent. We anticipate that the next generation of host-PCI bridges for processors with out-of-order instruction execution will support aggregating of PIO operations into bursts of 32 bytes or more, delivering 50% to 80% of bus bandwidth. Measurements with a Pamette PCI device [20] show that Digital's CIA bridge can burst consecutive PIO writes issued by the 21164 Alpha CPU on the PCI bus, for a bandwidth of 96 MB/s. In fact, Digital uses PIO exclusively for writing data to the MemoryChannel [15] network. To burst PIO on the receiving side, the processor must aggregate loads and present larger requests to the bridge. The 21164 CPU and CIA bridge can issue PIO reads in 32-byte bursts to deliver bandwidths of 47 MB/s

[2]Alcor/Miata, 8320-byte MTU, 1M netperf transfers, socket buffers at 1M, TCP checksums disabled on the receiver (hardware CRC only). This compares favorably with the 93 MB/s recently reported for a new reliable communication layer to replace TCP over VMMC-2 [13].

when quadword loads are issued in a 0-3-1-2 configuration for each line [20]. Given these capabilities, we believe that PIO is sufficient for all control message access.[3]

The Trapeze API includes a control message interface that insulates the caller from the details of how control message contents are moved to and from the adapter. Callers use *tpz_scan* and *tpz_reserve* primitives to obtain pointers into an incoming or outgoing message buffer at increasing offsets in a stack-like fashion. Depending on the Trapeze implementation for the platform, these pointers may directly address the NIC message buffers, or they may address an in-memory *shadow buffer*. The implementation may use shadow buffers to allow it to transfer the message contents to and from the NIC using DMA or PIO in whatever manner produces the best performance and stability on that specific platform. The cost of this technique is that control message data may pass through host memory.

3 Balancing DMA Latency and Bandwidth

This section deals with the handling of Trapeze payloads, which move to and from the host memory using DMA transfers on the PCI bus, under the control of the Trapeze firmware. We focus on the scheduling of transfers on the network link and host PCI buses so as to balance latency and bandwidth.

The Trapeze firmware uses a DMA pipelining technique called *adaptive cut-through delivery* to reduce latency of payloads larger than a few hundred bytes. In a recent paper [21], we described the mechanisms for a pure (non-adaptive) cut-through delivery scheme, measured its behavior for a range of packet sizes under different parameters on the Alcor platform, and extrapolated across a range of network link speeds and bus speeds. We briefly recap our approach and results in Section 3.1. We then explore the behavior of store-and-forward and cut-through variants across several platforms. In particular, we show the effect of cut-through delivery on point-to-point bandwidth, and show that good buffering and DMA transfer policies allow a cut-through scheme to adapt to deliver the bandwidth of store-and-forward under load.

3.1 Cut-Through Delivery

Cut-through delivery eliminates network adapter store-and-forward latency by pipelining packets through the adapter, similar to the way *cut-through routing* [18] eliminates store-and-forward latency in network switches. In Trapeze, cut-through delivery works for both incoming and outgoing packets as follows. The firmware initiates a new transfer on an outgoing DMA engine (received packet to the host or sending packet on the network link) whenever two conditions are satisfied. First, the DMA engine must be idle, as indicated by the LANai status bits. Second, the number of incoming bytes already deposited on the NIC (indicated by the LANai DMA counters) must exceed a configured DMA transfer size threshhold (*minpulse*).

[3]We measured these PIO bandwidths using a Pamette as the target device, but hardware bugs have prevented us from bursting PIO reads or writes with Myrinet on CIA-based systems. Myricom and Digital are actively diagnosing these problems with us. The Pyxis has the same PIO burst capabilities as the CIA, but all PIO bursting is disabled on the current Pyxis revision to improve stability of systems using the chip. As a result, CPU costs for control message access higher than expected in our current Trapeze prototype, but this has little effect on the performance results in this paper.

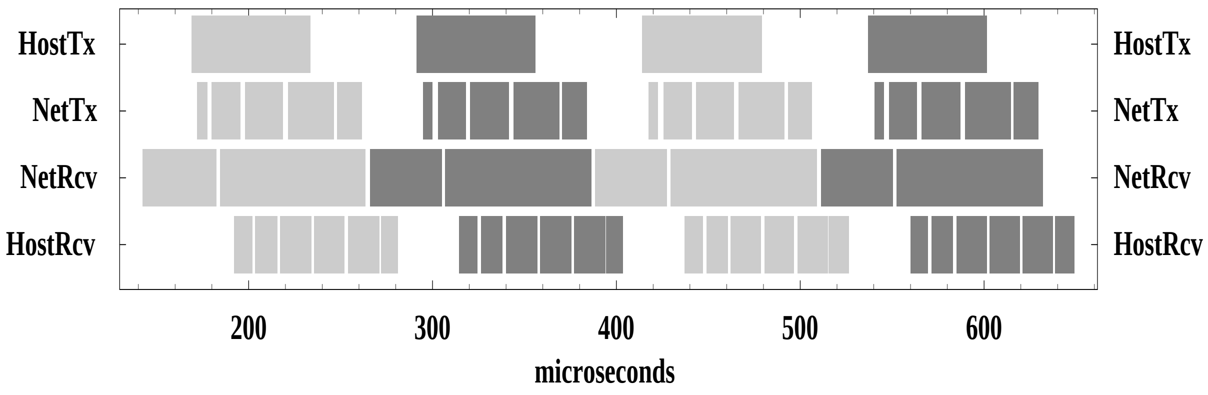

Figure 3: Effect of eager cut-through on 8K payload latency (P6/Natoma).

To prevent the DMA to the sink from overtaking the DMA from the source, outgoing transfers scheduled by the firmware never exceed the number of bytes already deposited on the NIC from the source. However, it is always beneficial for each transfer to include all of the bytes available from the sink at the time the transfer is initiated, rather than being limited to *minpulse*. We refer to this policy as *eager cut-through*. With eager cut-through, the lowest latency is achieved with the smallest *minpulse* threshholds (128 bytes) on both the sending and receiving sides.

3.2 Illustrating DMA Activity Graphically

Figure 3 shows the DMA transfers for a stream of 8K payloads transmitted between a pair of P6/Natoma systems using eager cut-through delivery. The packets are sent with a 130 μs inter-packet gap, enough to ensure that each packet clears the network before the next packet is sent. This *packet flow graph* and others in this paper were generated from logs of the DMA transfers taken by an instrumented version of the Trapeze firmware. The logs accumulate on the NIC memory for sampling intervals of a few milliseconds, and are then read by the host. Each log entry records the packet number, DMA engine ID, start time, completion time, and bytes transferred; the timings are taken from a cycle counter on the Myrinet NIC. The four horizontal bands show the transfers on the four DMA engines involved in transmitting a packet through the network, in order from top to bottom: sender's PCI (*hostTx*), sender's network channel (*netTx*), receiver's network channel (*netRcv*), and receiver's PCI (*hostRcv*).

The groups of vertical stripes for successive packets are represented with alternating shadings; all consecutive stripes with the same shading represent transfers of different fragments of the same packet. Payload transfers on the incoming DMA engines (*hostTx* and *netRcv*) are always issued for the full payload size, while cut-through delivery issues shorter transfers to the sink in order to overlap the DMA to the sink with DMA from the source. The only exception is the *netRcv* engine, which shows two transfers from the network channel for each incoming packet. The first of these transfers is for the 128-byte control message to which the subsequent payload is attached. Since the host reads and writes control message data using PIO, control message transfers do not appear in the *hostTx* and *hostRcv* bands. However, the entire 128-byte control message buffer is always transmitted on the network regardless of the amount of data it contains; thus *netTx* and *netRcv* begin each packet with a 128-byte transfer of the control message buffer contents from the sender's send ring to the

	4K Payloads		**8K Payloads**	
Systems	**Store/Forward**	**Cut-Through**	**Store/Forward**	**Cut-Through**
P6/Natoma	122μs, 99 MB/s	73μs, 71 MB/s	215μs, 111 MB/s	116μs, 85 MB/s
Pentium II/440LX	115μs, 99 MB/s	70μs, 72 MB/s	208μs, 113 MB/s	110μs, 86 MB/s
Alcor	158μs, 55 MB/s	109μs, 48 MB/s	282μs, 59 MB/s	177μs, 55 MB/s
Alcor to Miata	128μs, 91 MB/s	78μs, 70 MB/s	223μs, 106 MB/s	124μs, 82 MB/s

Table 1: One-way latency and bandwidth of 4K and 8K payloads under NIC DMA policies.

receiver's receive ring. This has a negligible effect on payload latency or bandwidth.

Note also that the width of each vertical stripe represents the duration of each transfer, not the number of bytes. The PCI has highest priority for LANai memory cycles, so the duration of *hostTx* and *hostRcv* transfers closely approximate their size, unless the bus arbiter preempts the transfer due to competition from another bus master (e.g., as for the first payload sent in Figure 4), or the bridge delays the transfer due to competition from the CPU for host memory bandwidth. However, the *netTx* and *netRcv* transfers for a packet occur in lockstep and may take longer than expected. For example, *netRcv* transfers stall until the sender transmits the data; these stalls are evident in Figure 3. Similarly, the *netTx* transfers do not complete until the receiving NIC has accepted all of the data; if a *netRcv* is not pending on the receiver, or if the receiving channel interface cannot consume the bytes fast enough, then the *netTx* is stalled by back-pressure flow control in the Myrinet link interfaces and switches. Moreover, both *netTx* and *netRcv* may be slowed by memory bandwidth limitations on the LANai.

3.3 Latency and Bandwidth of Eager Cut-Through

Figure 3 shows how eager cut-through delivery reduces packet latency relative to store-and-forward by hiding the latency of transfers on the host I/O bus, which is the bottleneck link for PCI implementations on the market today. By scheduling short transfers to the sink DMA engines as soon as data is available, cut-through delivery keeps idle DMA engines busy doing useful work. Note that each DMA transfer to the sink includes all of the bytes deposited in the adapter buffer by the source at the time the sink unit becomes available. Thus transfers to the sink tend to grow toward the end of the packet, as more data arrives from the source. This effect becomes more apparent in later figures.

However, pure cut-through delivery sacrifices bandwidth because the overhead of initiating each transfer to the sink is amortized across a smaller number of bytes. The bandwidth profiles in Figure 1 reflects the higher effective bandwidth for large transfers on the host PCI bus. The Myrinet network channel shows a similar relationship between transfer size and bandwidth.

Table 1 presents the measured latency and bandwidth for 4K and 8K transfers using pure cut-through delivery and store-and-forward on the Miata, Alcor, P6/Natoma, and Pentium-II/440LX platforms. A "pure cut-through delivery" system uses cut-through delivery as the only means to overlap transfers on the source and sink engines, whereas we assume double buffering on the adapter for store-and-forward, so the adapter may overlap transfer of a leading packet out of the adapter with transfer of a trailing packet into the adapter.

Table 1 shows that cut-through delivery cuts 8K packet latency almost in half on the Pentium-II platform, with improvements ranging from 32% to 46% for the other configurations. This can significantly reduce page fault stall times for GMS network memory applications on Myrinet clusters, our primary design goal. The expected latency benefit of cut-through delivery is highest for balanced buses close to the network link speed, thus it yields the highest benefit for the Pentium-II and the lowest benefit for Alcor, whose receive bandwidth is only half the link speed. However, pure cut-through delivery incurs a significant bandwidth penalty on all platforms.

Historically, LANs achieved only a fraction of the I/O bus bandwidth, and cut-through delivery would have a negligible effect on either large message latency or bandwidth. On low-end adapters, cut-through delivery is a good design choice on the basis of cost alone, since it reduces the need for buffer space on the NIC. However, when network speed is closely matched to the peak bandwidth of the I/O bus, the latency and bandwidth tradeoffs between cut-through and store-and-forward delivery become more pronounced. We expect this condition to hold for the forseeable future.

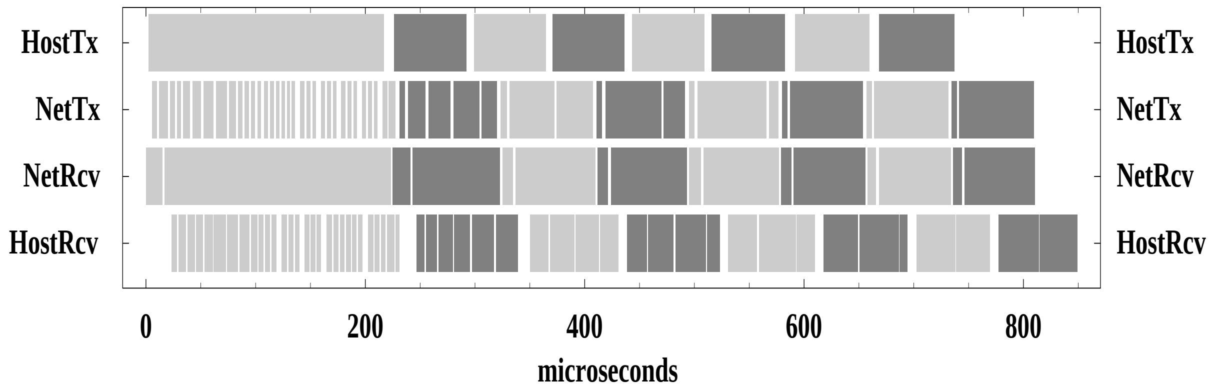

Figure 4: Adaptive cut-through delivery with a stream of 8K payloads (P6/Natoma).

3.4 Adaptive Cut-Through Delivery

The Trapeze firmware manages multiple buffers in NIC memory for moving payloads between the network and the host; it overlaps the transfer of a packet to a sink (*hostRcv* or *netTx*) with transfer of the next packet from a source (*hostTx* or *netRcv*). This is simply the well-known technique of double buffering, except that larger numbers of buffers are used to absorb bursts in the network traffic. This *multi-buffering* allows overlapped transfers of successive packets in a continuous stream, so the packets pipeline naturally in the network and adapters.

Multi-buffering combined with eager cut-through allows cut-through delivery to degrade naturally to store-and-forward under load, combining the low latency of cut-through delivery with the high bandwidth of store-and-forward. In general, the effect of cut-through for a bandwidth-constrained stream is that downstream links initially deliver lower effective bandwidth, since cut-through dictates that they will use smaller transfers. Since the source engines deposit data on the adapters faster than the downstream engines can sink it, data accumulates in the adapter buffers, priming the pipeline and triggering progressively larger transfers propagating downstream in a ripple effect. We call this combination of buffering and DMA policies *adaptive cut-through delivery*.

Figure 4 shows the DMA transfers at the start of a unidirectional bandwidth test with a stream of 8K payloads on a pair of P6-Natoma systems with *minpulse* set at 128 bytes. Both systems are idle at the start of the experiment. Consider first the behavior of the sender's transfers (*hostTx* and *netTx*). The host presents the first packet to Trapeze by storing the DMA address and length of the payload into the next available send ring entry. Since the entire 8K payload is available for DMA immediately, the firmware initiates a full 8K transfer on (*hostTx*). This transfer takes 214μs to complete due to competition from the host CPU as it writes to the send ring entries for subsequent packets. However, the *second* 8K payload transfer on *hostTx* runs at close to full bandwidth, completing in 66μs (124 MB/s). As bytes arrive on the adapter at full speed, the eager policy initiates progressively larger *netTx* transfers of 1384, 2420, and 2844 bytes, yielding link bandwidths of 90-121 MB/s.

On the receiving side, the *hostRcv* engine is idle when the second payload begins to arrive at T=218μs. The firmware initiates the first *hostRcv* transfer for 368 bytes at T=247μs. Due to the overhead of initiating such a small transfer, this transfer completes in 7μs for a bandwidth of 53 MB/s. However, while the transfer was in progress, 1012 more bytes arrived from the network on *netRcv*, causing the receiver to immediately initiate a second *hostRcv* transfer for 1012 bytes. Similarly, the third and subsequent transfers are even larger (1120, 1300, and 2072 bytes). Due to the use of multiple buffers, the increasing bus transfer sizes carry over to the next packet. Thus the receiver gradually drops out of cut-through as it comes under load and the amount of incoming data buffered on the adapter increases.

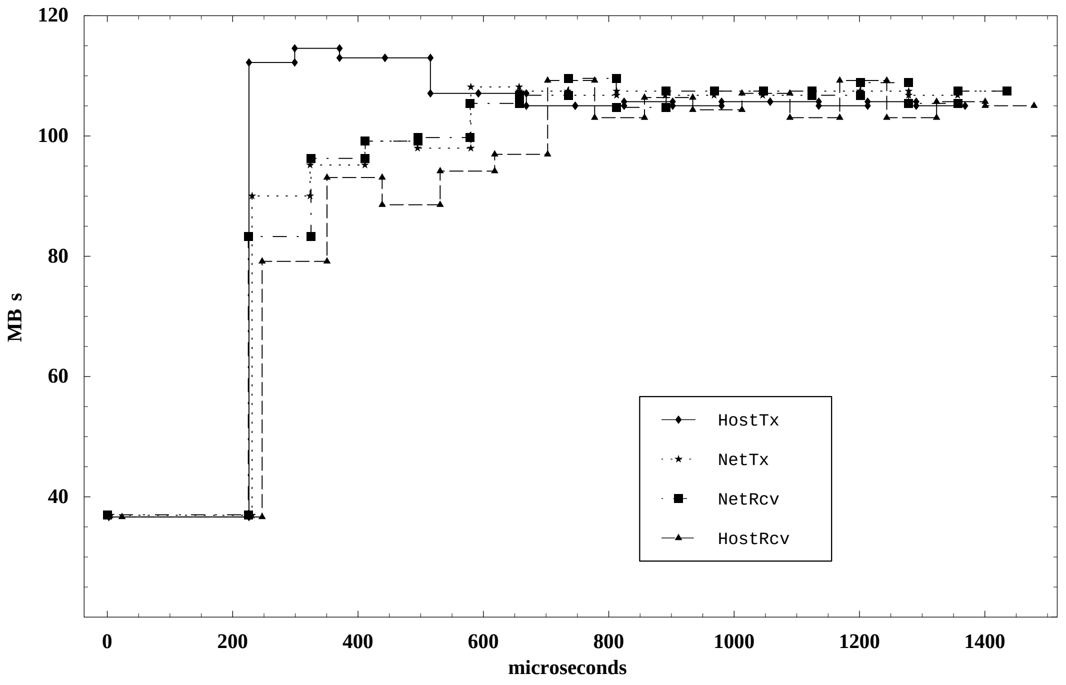

Figure 5: Stepping up effective link bandwidths under adaptive cut-through delivery.

Figure 5 shows the effective bandwidth through each DMA engine for the transfers at each point in time. These bandwidths include idle time between transfers, so they are slightly lower than the transfer bandwidths reported in the preceding paragraphs. The PCI send DMA (*hostTx*) spikes above 115 MB/s almost immediately, but the network channel bandwidth and receiver's PCI (*hostRcv*) bandwidth are "pulled up" to the sender's speed more gradually; the initial small transfers dictated by the cut-through policy deliver low bandwidth, which increases in steps as the eager DMA policy selects progressively larger transfers. The network channel ramps up first, followed by the receiver's PCI bus. This experiment reaches steady state at T=800μs, with all DMA engines transferring at an average bandwidth above 100 MB/s.

When running at full speed, about 15% of the bus bandwidth is lost due to messaging overheads and LANai occupancy, as shown by the gaps between transfers in the packet flow graphs. It is interesting to note that data is never presented to the *hostRcv* engine fast enough to force it out of cut-through completely. This is because the logging code costs enough drop bandwidth to a level that can be carried by 4K PCI transfers on the receiver's bus. With the logging compiled out, this experiment delivers 108 MB/s, forcing the *hostRcv* engine to fall back to 8K byte transfers.

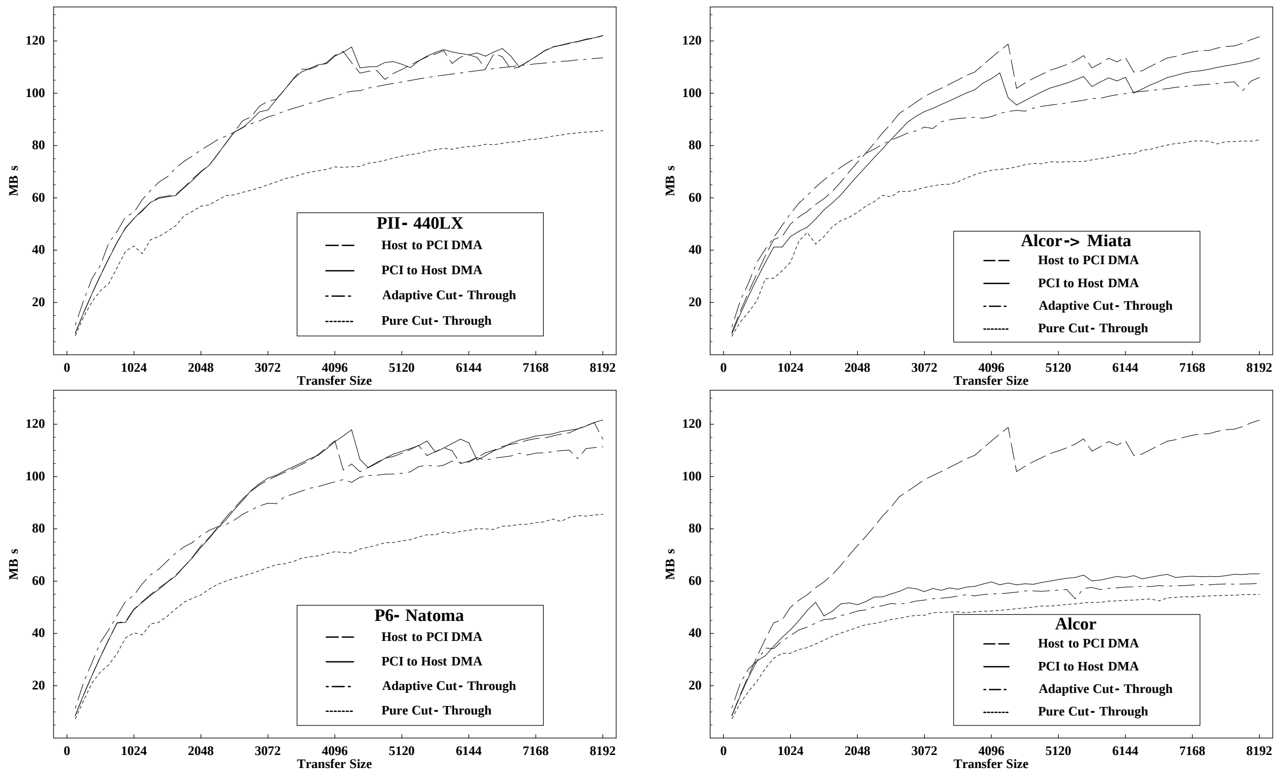

Figure 6: Bandwidth of NIC DMA policies for continuous streams of payloads of various sizes.

Figure 6 shows the measured bandwidth of Trapeze using pure cut-through delivery, store-and-forward, and adaptive cut-through delivery on the four workstation platforms, superimposed on the bandwidth profiles from Figure 1. These figures show that Trapeze with adaptive cut-through delivery closely matches the bandwidth of the bottleneck PCI channel on all the platforms, and perfectly matches the bandwidth of store-and-forward.

3.5 Effect of Cut-Through Policies on Trunk Bandwidth

We now briefly consider the effect of cut-through DMA policies on Myrinet trunk link bandwidth. Myrinet is a wormhole-routed network with link-level backpressure flow control. When a sender initiates a network packet transfer, the switch arbitration policies first acquire all trunk links and switch ports on the chosen path from the sender to the receiver; once the sender acquires these resources, it holds them until the entire packet has been transmitted or a watchdog timer expires and aborts the packet. By initiating *netTx* transfers early, cut-through policies may hold the network link open for the entire time it takes to transfer the last $payloadsize - minpulse$ bytes of the packet to the NIC over the PCI bus. Although the sending NIC transmits the payload on the network link as a sequence of *netTx* transfes, all of these transfers appear to the network as a single packet transmission, effectively throttling the 160 MB/s network link to the delivered bandwidth of the sender's PCI.

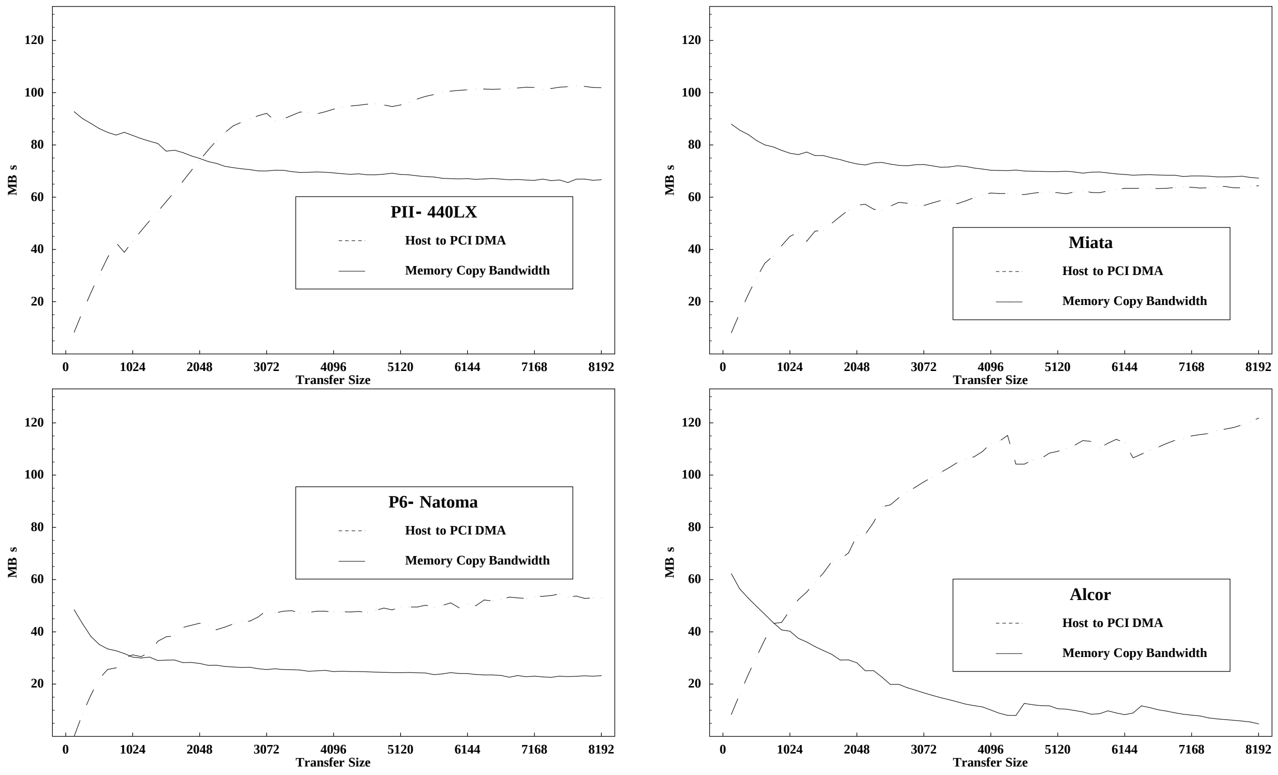

Figure 7: Delivered bandwidth of concurrent block copy and DMA read (send) operations on four workstation platforms.

Host-PCI bridges vary significantly in their handling of competing requests. Figure 7 shows the allocation of memory system bandwidth between the host CPU and the PCI by all four bridges when faced with DMA read requests from a bus master (Myrinet *hostTx*) while the CPU is executing block copy operations. The Natoma bridge balances memory system bandwidth between the CPU and the DMA transfer, which has the effect of slowing down the send DMA by almost 70%. The Alpha CIA bridge starves the CPU in favor of a DMA read, so it does not slow transfers to the NIC. The 440LX and Pyxis bridges deliver almost sufficient memory system bandwidth to handle both transfers simultaneously, minimizing the problem.

The adaptive nature of cut-through delivery will minimize this problem in practice. If there is no competing traffic to the receiver or through a common trunk link, then there is no effect on performance. If there is competing traffic then some of each sender's *netTx* transfers will stall to acquire resources held by another sender. During the stall, bytes from the *hostTx* transfer will accumulate on the adapter, pushing the *netTx* engine out of cut-through on the next transfer.

4 Conclusion

Designers of I/O devices — particularly high-speed network adapters — face a range of choices for handling the movement of data to and from host memory. These choices have significant effects on performance for common network traffic patterns in which large-message latency and bandwidth are important. In addition, the characteristics of the host I/O bus can significantly affect the tradeoffs.

This paper examines the behavior of DMA and buffering policies used by a high-speed network adapter connected to the host through a PCI I/O bus, with particular focus on the behavior of cut-through DMA policies that pipeline packets through the adapter. We define and evaluate a DMA policy that combines double buffering with eager cut-through DMA to create an adaptive cut-through delivery hybrid, and report on experiences with high-speed network communication using the Trapeze messaging system on four workstation platforms based on host-PCI bridges with different characteristics. The paper shows that adaptive cut-through delivery yields low packet latencies while degrading naturally to a store-and-forward policy for high bandwidth under load. Moreover, adaptive cut-through delivery falls back to larger, more efficient transfers when faced with a high-bandwidth source channel or competition for bandwidth to a sink, which is important on Myrinet and other wormhole-routed networks.

References

[1] Darrell C. Anderson, Jeffrey S. Chase, Syam Gadde, Andrew J. Gallatin, Kenneth G. Yocum, and Michael J. Feeley. Cheating the I/O bottleneck: Network storage with Trapeze/Myrinet. In *submitted for publication*, 1997.

[2] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[3] Matthias A. Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki, Edward W. Felten, and Jonathon Sandberg. Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153, April 1994.

[4] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W-K Su. Myrinet - a gigabit-per-second local area network. *IEEE Micro*, February 1995.

[5] José Carlos Brustoloni and Peter Steenkiste. Effects of buffering semantics on I/O performance. In *Proc. of the Second Symposium on Operating Systems Design and Implementation*, pages 277–291, Seattle, WA, October 1996. USENIX Assoc.

[6] Greg Buzzard, David Jacobson, Milon Mackey, Scott Marovich, and John Wilkes. An implementation of the Hamlyn sender-managed interface architecture. In *Proc. of the Second Symposium on Operating Systems Design and Implementation*, pages 245–259, Seattle, WA, October 1996. USENIX Assoc.

[7] Jeff Chase, Andrew Gallatin, Alvin Lebeck, and Ken Yokum. *Trapeze API.* Duke University, 1997.

[8] Brent N. Chun, Alan M. Mainwaring, and David E. Culler. Virtual network transport protocols for Myrinet. In *Hot Interconnects Symposium V*, August 1997.

[9] David E. Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun, Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, and Frederick Wong. Parallel computing on the Berkeley NOW. In *Proceedings of the 9th Joint Symposium on Parallel Processing (JSPP 97)*, 1997.

[10] Zubin D. Dittia, Guru M. Parulkar, and Jerome R. Cox. The APIC approach to high performance network interface design: Protected DMA and other techniques. In *Proceedings of IEEE Infocom*, 1997. WUCS-96-12 technical report.

[11] Peter Druschel, Mark B. Abbott, Michael Pagels, and Larry L. Peterson. Network subsystem design. *IEEE Network (Special Issue on End-System Support for High Speed Networks)*, 7(4):8–17, July 1993.

[12] Peter Druschel, Larry L. Peterson, and Bruce S. Davie. Experience with a high-speed network adaptor: A software perspective. In *Proceedings of the SIGCOMM '94 Symposium*, August 1994.

[13] Cezary Dubnicki, Angelos Bilas, Yuqun Chen, Stefanos Damianakis, and Kai Li. VMMC-2: Efficient support for reliable, connection-oriented communication. In *Hot Interconnects Symposium V*, August 1997.

[14] Michael J. Feeley, William E. Morgan, Frederic H. Pighin, Anna R. Karlin, and Henry M. Levy. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[15] Marco Fillo and Richard B. Gillett. Architecture and implementation of MEMORY CHANNEL2. *Digital Technical Journal*, 9(1), 1997.

[16] PCI Special Interest Group. *PCI Local Bus Specification 2.1.* 1995.

[17] Hsiao-Keng and Jerry Chu. Zero-copy TCP in Solaris. In *Proceedings of the USENIX 1996 Annual Technical Conference*, January 1996.

[18] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks 3, pages 267-286, 1979.*, 3:267–286, 1979.

[19] Richard P. Martin, Amin M. Vahdat, David E. Culler, and Thomas E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 85–97, June 1997.

[20] Laurent Moll and Mark Shand. System performance measurement on PCI Pamette. In *Proceedings of the Symposium on Field-Programmable Custom Computing Machines (FCCM 97)*, 1997. Digital Equipment Corporation Systems Research Center.

[21] Omitted. Cut-through delivery in Trapeze: An exercise in fast messaging. 1997.

[22] Scott Pakin, Vijay Karamcheti, and Andrew Chien. Fast Messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Parallel and Distributed Technology*, 1997.

[23] Charles L. Seitz, Nanette J. Boden, Jakov Seizovic, and Wen-King Su. The design of the Caltech Mosaic C multicomputer. In *Proceedings of the University of Washington Symposium on Integrated Systems*. MIT Press, 1993.

[24] Chandramohan A. Thekkath and Henry M. Levy. Limits to low-latency communication on high-speed networks. *ACM Transactions on Computer Systems*, 11(2), May 1993.

[25] Matt Welsh, Anindya Basu, and Thorsten von Eicken. Incorporating memory management into user-level network interfaces. In *Hot Interconnects Symposium V*, August 1997.