High Performance Messaging on Workstations:
Illinois Fast Messages (FM) for Myrinet*

Scott Pakin† Mario Lauria‡ Andrew Chien†

Abstract

In most computer systems, software overhead dominates the cost of messaging, reducing delivered performance, especially for short messages. Efficient software messaging layers are needed to deliver the hardware performance to the application level and to support tightly-coupled workstation clusters.

Illinois Fast Messages (FM) 1.0 is a high speed messaging layer that delivers low latency and high bandwidth for short messages. For 128-byte packets, FM achieves bandwidths of 16.2 MB/s and one-way latencies 32 μs on Myrinet-connected SPARCstations (user-level to user-level). For shorter packets, we have measured one-way latencies of 25 μs, and for larger packets, bandwidth as high as to 19.6 MB/s — **delivered** bandwidth greater than OC-3. FM is also superior to the Myrinet API messaging layer, not just in terms of latency and usable bandwidth, but also in terms of the message half-power point ($n_{\frac{1}{2}}$), which is two orders of magnitude smaller (54 vs. 4,409 bytes).

We describe the FM messaging primitives and the critical design issues in building a low-latency messaging layers for workstation clusters. Several issues are critical: the division of labor between host and network coprocessor, management of the input/output (I/O) bus, and buffer management. To achieve high performance, messaging layers should assign as much functionality as possible to the host. If the network interface has DMA capability, the I/O bus should be used asymmetrically, with

*The research described in this paper was supported in part by NSF grants CCR-9209336 and MIP-92-23732, ONR grants N00014-92-J-1961 and N00014-93-1-1086 and NASA grant NAG 1-613. Andrew Chien is supported in part by NSF Young Investigator Award CCR-94-57809.

†Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Ave., Urbana, IL 61801, USA

‡Dipartimento di Informatica e Sistemistica, Università di Napoli “Federico II”, via Claudio 21, 80125 Napoli, Italy

the host processor moving data to the network and exploiting DMA to move data to the host. Finally, buffer management should be extremely simple in the network coprocessor and match queue structures between the network coprocessor and host memory. Detailed measurements show how each of these features contribute to high performance.

1 Introduction

As the performance of workstations reaches hundreds of megaflops (even gigaflops), networks of workstations provide an increasingly attractive vehicle for high performance computation [3]. In fact, workstation clusters have a number of advantages over their major competitors (massively-parallel processors based on workstation processors). These advantages can include lower cost, a larger software base, and greater accessibility. Further, the advent of high performance network interconnects such as ATM [7], Fibre Channel [4], FDDI [13], and Myrinet [6] present the possibility that workstation clusters can deliver good performance on a broader range of parallel computations.

Achieving efficient communication is the major challenge in synthesizing effective parallel machines from networks of workstations. Unfortunately, to date the most common messaging layers used for clusters (TCP/IP [9], PVM [27]) generally have not delivered a large fraction of the underlying communication hardware performance to the applications. Reasons for this include protocol overhead, buffer management, link management, and operating system overhead. Even in recent high speed network experiments, high bandwidths are generally only achieved for large messages (hundreds of kilobytes or even megabytes) and then only with overheads of 1 millisecond or more. Reasons for this include system call overhead, buffer copying, network admission control, poor network management, and software overhead . As a result, parallel computing on workstation clusters has largely been limited to coarse-grained applications.

Attempts to improve performance based on specialized hardware can achieve dramatically higher performance, but generally require specialized components and interfacing deep into a computer system design [16, 18, 19]. This increases cost, and decreases the potential market (and hence sale volume) of the network hardware.

The goal of the Illinois Fast Messages (FM) project is to deliver a large fraction of the network's physical performance (latency and bandwidth) to the user at small packet sizes.[1] Building efficient soft-

[1] More information and software releases of FM are available from: http://www-csag.cs.uiuc.edu/projects/communication/sw-messaging.html .

ware messaging layers is not a unique goal [12, 25, 30, 31], but FM is distinguished by its hardware context (Myrinet) and high performance.

The Fast Messages project focuses on optimizing the software messaging layer that resides between lower-level communication services and the hardware. It is available on both the Cray T3D [22, 23] and Myricom's Myrinet [6]. Using the Myrinet, FM provides MPP-like communication performance on workstation clusters. FM on the Myrinet achieves low-latency, high-bandwidth messaging for short messages delivering 32 μs latency and 16 MBytes/s bandwidth for 128 byte packets (user-level to user-level). For shorter packets, latency drops to 25 μs, and for larger packets, bandwidth rises to 19.6 MB/s. This **delivered** bandwidth is greater than OC-3 ATM's physical link bandwidth of 19.4 MB/s. FM's performance exceeds the messaging performance of commercial messaging layers on numerous massively-parallel machines [21, 29, 11]. A good characterization of a messaging layer's usable bandwidth (bandwidth for short messages) is $n_{\frac{1}{2}}$, the packet size to achieve half of the peak bandwidth ($\frac{r_\infty}{2}$). FM achieves an $n_{\frac{1}{2}}$ of 54 bytes. In comparison, Myricom's commercial API requires messages of over 3,873 bytes to achieve the same bandwidth. FM has improved the network's ability to deliver performance to short messages dramatically, reducing $n_{\frac{1}{2}}$ by nearly two orders of magnitude.

In the design of FM, we addressed three critical design issues faced by all designers of input/output bus interfaced high speed networks: division of labor between host and network coprocessor, management of the input/output bus, and buffer management. To achieve high performance, messaging layers should assign as much functionality as possible to the host. This leaves the network coprocessor free to service the high speed network channels. If the network interface has DMA capability, the input/output bus should be used asymmetrically, with the host processor moving data to the network and exploiting DMA to move data to the host. Using the processor to move data to the network reduces latency, particularly for small messages. DMA transfer for incoming messages, initiated by the network coprocessor, maximizes receive bandwidth with little cost in latency. Finally, buffer management should be extremely simple in the network coprocessor and match queue structures between the network coprocessor and host memory. Simple buffer management minimizes software overhead in the network coprocessor, again freeing the coprocessor to service the fast network. Matching queue structures between the host and network coprocessor allows short messages to be aggregated in DMA operations, reducing the data movement overhead. Detailed measurements evaluate several design alternatives and show how

each of these achieves high performance.

The rest of the paper is organized as follows. Section 2 describes issues common to all messaging layers. Section 3 explains our FM design in light of the hardware constraints of a workstation cluster. In Section 4, we present the design and performance of FM elements in detail, justifying each design decision with empirical studies. We discuss our findings in Section 5 and provide a brief summary of the paper and conclusions in Section 6.

2 Background

For some time, researchers and even production sites have been using workstation clusters for parallel computation. Many libraries are available to support such distributed parallel computing (PVM [27] atop UDP or TCP [28] is perhaps the most popular). The communication primitives in these libraries have typically exploited operating system communication services, running atop 10 Mb/s Ethernet, or more recently some higher speed physical media such as FDDI [13], ATM [7] or Fibre Channel [4]. While such facilities are useful for coarse-grain decoupled parallelism, they suffer from high software communication overhead (operating system calls) and low achieved bandwidth (media limits or software overhead), and thus cannot support more tightly coupled or finer-grained parallelism.

Higher performance messaging systems for workstation clusters often bypass the operating system, mapping the network device interface directly into the user address space and accessing it directly via load/store operations. Protection can still be achieved at virtual address translation but sharing of communication resources is more complicated. Our FM layer uses this approach, mapping the Myrinet network interface directly into the user address space. Note that even with a memory-mapped interface accesses can still be expensive; in our Myrinet system, reading a network interface status field requires ≈ 15 processor cycles. Some ATM systems provide memory mapped input/output bus interfaces, but achieving performance is still a challenging proposition. For example, delivered bandwidths of 1–3 MB/s are typical [24]. Achieving high performance requires careful management of the hardware resources by the software messaging layer.

How deeply network interfaces will be integrated into a typical system is a debate currently raging in the workstation cluster community. Less integrated solutions are favored by third party network providers, giving leverage for their designs over many systems. More tightly integrated designs clearly make it easier to achieve low latency and high bandwidth. The Myrinet hardware used for this work rep-

resents the former approach, interfacing the network to the Sun's input/output bus (SBus). Consequently, many parts of the workstation architecture affect performance; in the following paragraphs we describe the performance of these salient network and workstation features.

Myrinet Network Features

Myrinet is a high speed LAN interconnect which uses byte-wide parallel copper links to achieve physical link bandwidth of 76.3 MB/s [6]. Myrinet uses a network coprocessor (LANai) which controls the physical link and contains three DMA engines (incoming channel, outgoing channel, and host) to move data efficiently. Host-LANai coordination is achieved by mapping the LANai's memory into the host address space. Though the LANai integrates much of the critical functionality (low-overhead DMA), the current version (2.3) is a rather slow processor, a CISC architecture operating at the SBus clock frequency (20–25 Mhz) and executing one instruction every 3–4 cycles. The low speed of the LANai processor ($\approx$ 5 MIPS) compared to the network makes LANai program design critical in achieving high performance. For example, spooling a packet of 128 bytes over the channel takes 1.6 μs, the equivalent of only about eight to ten LANai instructions!

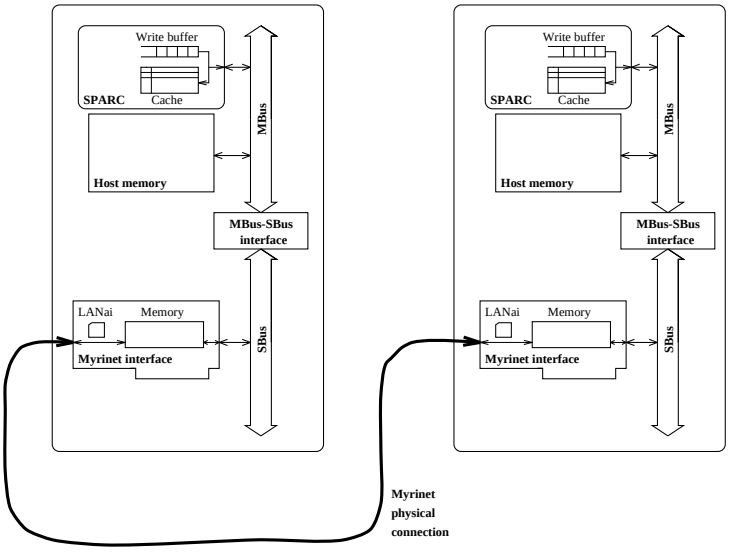


Figure 1: Host-to-host data path

Workstation Features

Critical issues in the workstation include the processor speed, memory performance (memory bus) and input/output performance

(input/output bus). While latency is important, for longer messages the critical issue is bandwidth. To reach the network, data must first traverse the processor memory hierarchy (write buffer, memory bus), then the input/output bus (see Figure 1). Because the memory bus (MBus) has typically much better performance, the input/output bus (SBus) is generally the bottleneck. SBus performance favors DMA, supporting it with special burst-mode transfers which provide 40–54 MB/s for large transfers. With the Myrinet network interface, DMA can only initiated by the LANai and transfers must be to or from kernel memory. For processor mediated transfers, using double-word writes achieves a maximum of 23.9 MB/s.

We use two workstations for our measurements: a SPARCstation 20 with two 50 MHz SuperSPARC processors (without the optional L2 cache) and a SPARCstation 10 with four 55 MHz RT100 HyperSPARCs. The multiplicity of processors is not a significant issue. The memory bandwidths of the processors are 60 MB/s writes, 80 MB/s reads and 52 MB/s writes, 37 MB/s reads respectively. These memory bandwidths are greater than the processor-mediated SBus bandwidth, and hence are not a critical performance factors.

3 The Fast Messages Approach

3.1 Illinois Fast Messages (FM) 1.0

Illinois Fast Messages (FM) is a high performance messaging layer which is available on several parallel platforms (Cray T3D and workstation clusters) [22, 23]. The design goal of FM is to deliver network hardware performance to the application level with a simple interface. FM is appropriate for implementors of compilers, language runtimes, communications libraries, and in some cases application programmers.

Function	Operation
FM_send_4(dest,handler,i0,i1,i2,i3)	Send a four word message
FM_send(dest,handler,buf,size)	Send a long message
FM_extract()	Process received messages

Table 1: FM 1.0 layer calls

Table 1 lists all three of FM's messaging functions. There are two calls to send messages, `FM_send_4()` and `FM_send()`, for extremely short, and somewhat longer (32 words or fewer) messages. Each message carries a pointer to a sender-specified function (called a

"handler") that consumes the data at the destination. The handler-carrying message concept is similar to Active Messages [31], but in FM there is no notion of request-reply coupling. There are no restrictions on the actions that can be performed by an handler, and it is left to the programmer of preventing deadlock situations. When a process wishes to check for and process a message, it calls `FM_extract()`, which dequeues and processes one or more messages. Because host processor involvement is not required to remove data from the network, polling is not required to prevent network blockage. Similar to Active Messages, message buffers do not persist beyond the return of the handler.

3.2 Critical Messaging Layer Issues

The primary purpose of a messaging layer is the efficient transport of data from one processor to another. Messaging layers hide the underlying hardware and software components, providing services such as reliable delivery, synchronization, in-order delivery, and collective communication in addition to basic data movement. We are concerned with minimal messaging layers, so FM includes only features whose omission would cause major performance losses if handled in higher software layers.

The basic feature provided by FM is reliable delivery which is deemed necessary due to the costs of source buffering, timeout, and retry in higher software layers. Reliable delivery alone requires the messaging implementation to resolve issues of flow control, buffer management, and the division of labor between the host processor and the network coprocessor. Messaging layers generally provide flow control to avoid data loss and therefore, data retransmission. Flow control is required because all real computers and networks have only a finite amount of buffering—flow control prevents buffer overflow. Optimal flow control matches send rates to receive rates with minimal overhead.

Buffer management enables the reuse of the finite host and network coprocessor memories to handle incoming and outgoing messages. Buffering is employed to match rates between the host processor, network interface, and network channels. Buffers support different processing rates and service intervals, decoupling network, LANai, and host. An ideal buffering scheme would allow buffer allocation and release in any order at minimal overhead.

The division of labor between the host processor and the network coprocessor is a key performance issue in any system with a programmable network coprocessor. While most services can be programmed on either the host or the network interface, balanced decompositions allow overlapping of the two levels and supporting

a higher message rate. Of course, configurations of host and network coprocessor in which the host processor is much faster and has significantly more memory [6, 20, 26] (e.g. our configuration) favor assigning more work to the host.

4 Fast Messages 1.0 Implementation Design

The FM 1.0 implementation consists of two basic parts: the host program and the LANai control program (LCP). These programs coordinate through the LANai memory which is mapped into the host processor's address space, but rather expensive to access because it resides on the SBus. Thus, a message transmission consists of the following steps: getting the data to the LANai (traversing the sender's memory bus and input/output bus), putting the data onto the communication channel, removing it from the communication channel at the receiver, and transporting the data to the receiver host's memory. Each of these steps contributes to communication latency, and the slowest of them determines the maximum sustainable bandwidth. Thus, all parts of the system—both hardware and software—can be critical. Exploiting the Myrinet's high speed links effectively requires careful design of both the host and LCP both to be efficient and to manage the bandwidth of the SBus and physical channel effectively.

Critical issues in the design of the FM messaging layer include the structure of the LCP, buffer management and coordination of the two programs across the SBus, and the design of the host program. Careful design of all parts contributes to producing a high performance messaging layer. To elucidate the contribution of each factor to performance, we examine each in turn, building up from a minimal network coprocessor program that merely sends data across the channel (never getting it to the hosts) to a complete messaging layer.

4.1 Performance Metrics and Measurements

To characterize communication performance, we use several standard parameters (see Table 2). r_∞ bounds the maximum possible throughput, and t_0 captures the leanness of the implementation. Since our goal is to support short messages more effectively, we use $n_{\frac{1}{2}}$ to show how successful we are in delivering good network performance for short messages. These performance metrics are calculated from measurements of latency and throughput.

Network latency is measured by ping-ponging a message back and forth 50 times, and dividing to compute the one-way packet latency. Bandwidth is determined by measuring the time to send 65,535 packets and dividing the volume of data transmitted by the elapsed

Metric	Definition
r_∞	Peak bandwidth for infinitely large packets (asymptotic)
$n_{\frac{1}{2}}$	Packet size to achieve bandwidth of $\frac{r_\infty}{2}$
t_0	Startup overhead
ℓ	Packet latency (one way)

Table 2: Definitions of performance metrics

time. All measurements were taken on an 8-port Myrinet switch and a pair of workstations (see Section 2). In all of our measurements, message length refers to the payload (so that the reported data are inclusive of the header overhead), and 1 MB $\equiv$ 2^{20} bytes.

4.2 Network Coprocessor Program

The network coprocessor program is a critical contributor to performance, incurring latency and bounding the peak bandwidth achievable. Because the network coprocessor (LANai) is of modest speed, and the LANai control program (LCP) is a sequential program dealing with concurrent activities, the organization of the LCP is critical to achieving high performance. We consider two basic implementations of the LANai control program's main loop. The first, **baseline**, is the straightforward logical structure shown in Figure 2(a). The second version of the LCP loop, **streamed**, optimizes performance by consolidating checks for queue management and by streaming sends and receives to improve peak performance (see Figure 2(b)). As computer traffic is often quite bursty, streaming is likely to improve average performance as well.

To calibrate our results and show how the LANai's speed can impact network performance, we compare performance for the **baseline** and **streamed** performance against the LANai's theoretical peak performance. Theoretical peak performance is indicated as **theoretical peak** and is calculated for an LCP which does DMAs of the appropriate size, omitting any pointer updates, checks for completion, queue boundary checks, looping overhead, etc. (see Appendix A).

The main loop organization in the LCP is a critical contributor to both latency and bandwidth for short messages. As shown in Figure 3, **baseline** incurs a latency far greater than the theoretical minimum, indicating that even mundane pointer and looping overheads reduce performance significantly. The **Baseline** loop achieves a $t_0 = 4.2\,\mu$s, and $n_{\frac{1}{2}} = 315$ bytes. **Streamed** improves the basic LCP loop, achieving higher bandwidth and lower latency, especially

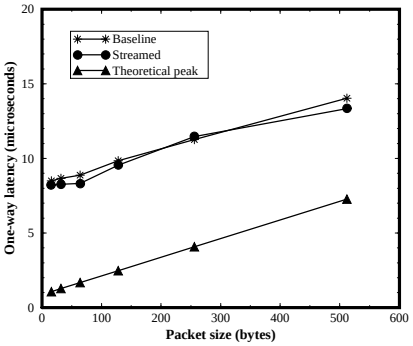
```
PACKET   sendbuffer
PACKET   receivebuffer
integer  hostsent      /* Total  # of msgs. host wants to send */
integer  lanaisent     /* Total  # of msgs. LCP has sent */

repeat forever
   if hostsent != lanaisent and send channel is available then
      send packet from a fixed buffer location
      lanaisent++
   end if
   if a packet is available on the receive channel then
      receive packet into a fixed buffer location
   end if
end repeat
```

(a) Baseline

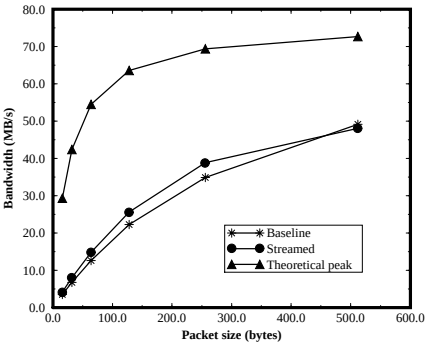
```
PACKET   sendbuffer
PACKET   receivebuffer
integer  hostsent      /* Total  # of msgs. host wants to send */
integer  lanaisent     /* Total  # of msgs. LCP has sent */

repeat forever
   while hostsent != lanaisent and send channel is available then
      send packet from a fixed buffer location
      lanaisent++
   end while
   while a packet is available on the receive channel then
      receive packet into a fixed buffer location
   end while
end repeat
```

(b) Streamed

Figure 2: Pseudocode for the LCP main loop



(a) Latency



(b) Bandwidth

Figure 3: LANai to LANai Performance

for short messages. More specifically, **streamed** achieves performance of $t_0 = 3.5\,\mu$s and $n_{\frac{1}{2}} = 249$ bytes. In both case, r_∞ is 76.3 MB/s, the maximum link bandwidth.[2] Even without any host processor or SBus involvement, a minimal LANai control program incurs significant startup overhead, producing latencies much higher than the theoretical peak. Both versions of the LCP can achieve full link bandwidth, but they require large messages to do so. In all cases, the **streamed** version is significantly better, so we build on the **streamed** LCP loop from this point forward.

4.3 SBus Management

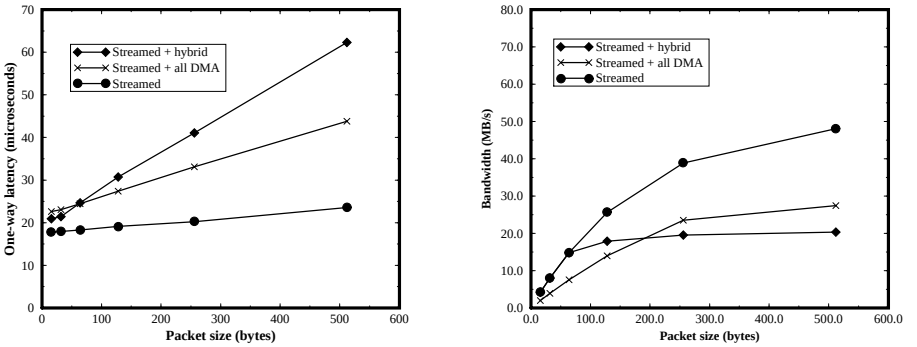
For high speed networks that interface to the input/output bus, the speed and latency of that bus are critical performance constraints. Interaction costs between host and network coprocessor via the I/O bus determine the feasible architectures for software messaging layers. However, because I/O busses are widely standardized, they form a cost-effective interface for networking hardware vendors. Consequently, they are the most common level of interface for high speed networks. The SPARCstation's SBus can achieve high bandwidth only for DMA transfers (see Section 2)—a significant performance constraint.

We consider two possible architectures for interaction between the host and LANai: **all-DMA** and **hybrid**. The first, **all-DMA**, attempts to maximize bandwidth by using DMA to move data both to and from the network. For outgoing messages, the host copies data into the DMA region, writes message pointers to the LANai, and triggers the send. The copy to the DMA region is necessary because DMA operations can occur only between a device and a pinned-down, kernel-mapped DMA region. For incoming messages, the host writes a buffer pointer to the LANai, and the LANai uses DMA to transfer the message into host memory. The second, **hybrid**, uses the host to move data directly to the LANai's memory and triggers the send (both over the SBus). This avoids the memory to memory copy and eliminates one synchronization between host and LANai. For incoming messages, the LCP simply DMAs messages into the host memory.

To compare performance, we measured latency and bandwidth for a range of packet sizes, layering these vestigial host programs atop the superior **streamed** LCP main loop. Time is measured from the `FM_send()` call until the (essentially empty) handler returns. In all cases, data copying is achieved with a memory-to-memory copy

[2] At present, we cannot explain the crossover points for latency (256 byte packets) and bandwidth (512 byte packets). We are trying to track down the source of these perturbations.

function optimized (opencoded) for the packet size.



(a) Latency (b) Bandwidth

Figure 4: Minimal host to host performance

Because the cost of interaction and data movement across the SBus is high, extending the messaging layer out to the hosts produces dramatically lower performance. Both protocols for managing the SBus produce much higher latencies for all message sizes, and saturate at lower bandwidths than the underlying **streamed** LCP (see Figure 4). **all-DMA** incurs a large additional latency because of a memory to memory copy, and two synchronizations for SBus operations. However, using DMA transfers delivers high SBus bandwidth, so **all-DMA** achieves $t_0 = 7.5\,\mu$s, $r_\infty = 33.0$ MB/s and $n_{\frac{1}{2}} = 162$ bytes. In contrast, **hybrid** incurs less additional latency initially, by avoiding the memory to memory copy, and requiring only one synchronization. However, because processor-mediated data movement achieves lower SBus bandwidth, the SBus becomes the performance limitation. The 21.2 MB/s peak bandwidth approaches the maximum write bandwidth on the SBus. Overall, **hybrid** achieves $t_0 = 3.5\mu$s, $r_\infty = 21.2$ MB/s, and $n_{\frac{1}{2}} = 44$ bytes.

The poor performance of processor mediated data movement forces a performance tradeoff between short and long message performance. Incurring latency degrades short message performance, but delivers high bandwidth, improving long message performance. To optimize short message performance in FM, we choose to use the **hybrid** scheme. This choice can increase the host overhead for messaging. Improved I/O bus bridges and higher performance I/O busses could eliminate this tradeoff, enabling both high bandwidth and low latency communication. Even the latency-optimized code

Characteristic	Reg. Mem.	DMA region	LANai
Capacity	Virtual memory	Physical memory	128 KB
Host access	Loads/store	Loads/store	Load/store
LANai access	none	DMA only	Load/store

Figure 5: Memory characteristics

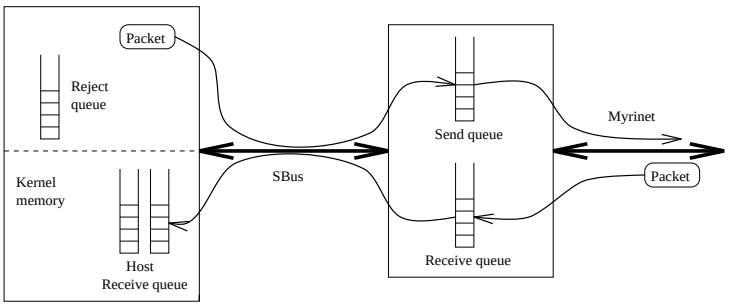


Figure 6: Host and LANai queues

produces rather high absolute bandwidth, but we still must add buffer management and flow control to make the messaging layer useful.

4.4 Buffer Management

Buffer management is a critical issue for messaging layers, as it often accounts for remarkably large fractions of the messaging overhead. Our minimally-functional messaging layer thus far assumes infinite buffering; any useful messaging layer must recycle its storage. In addition, since the host and LANai must share message queues, efficient synchronization is critical. Data can be buffered in three locations: LANai memory, host DMA region, or regular host memory. Each type of memory has different capacity and accessibility characteristics (see Table 5).

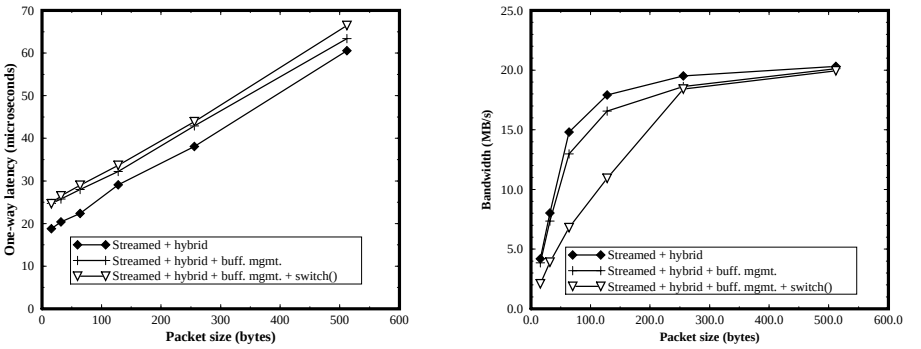
Because buffer management can be expensive, FM is designed to use only four queues: LANai send, LANai receive, host receive, and host reject queue (see Figure 6). More complex structures were eschewed because even minimal multiplexing amongst queues in the LANai reduces performance dramatically, and the cost of sharing queues between the host and LANai is significant.

Outgoing packets are copied by the host directly into the LANai send queue, and the packet's presence is triggered by updating the `hostsent` counter in the LANai memory. The LCP then DMAs the packet out to the network channel. To avoid write races on memory locations, the LANai uses a separate counter to keep track of the

LANai send queue (the `lanaisent` counter) which always trails the `hostsent` counter by the number of packets in the queue. Allowing each to own (and keep in a register) its respective counter reduces the amount of synchronization between host and LANai.

For incoming packets, the LCP first DMAs the packet from the network into the LANai receive queue. When the SBus becomes available, the LCP DMAs all undelivered packets to the host memory. The LANai does no interpretation of packets, blindly moving them to the DMA region. Using the LANai to move packets to the host frees the host to perform other computations but also enables higher SBus bandwidths through the use of DMA. The bandwidth issue is critical, as delivering incoming packets to the host is often the critical bottleneck in high-performance networks,

We chose to do no packet interpretation in the LANai for several reasons. First, the cost of interpretation in the LANai is significant (see Figure 7). Second, having no packet interpretation and a simple LANai receive queue structure allows packets to be aggregated and transferred with a single DMA operation, further increasing the transfer bandwidth and reducing overhead. This simple LCP leaves packet interpretation and sorting to the host (separating incoming packets from rejected packets).



(a) Latency (b) Bandwidth

Figure 7: Host to Host performance with buffer management

Adding FM's streamlined buffer management incurs some latency, but preserves nearly all of the bandwidth of the messaging layer. Figure 7 shows the performance of the **hybrid** layer, the **hybrid** layer augmented with buffer management, and the **hybrid** layer augmented by both buffer management and a `switch()` state-

ment to simulate packet interpretation. Compared to the **hybrid** layer's performance, $t_0 = 3.5\,\mu$s, $r_\infty = 21.2$ MB/s, and $n_{\frac{1}{2}} = 44$ bytes, the **hybrid + buffer management** layer achieves $t_0 = 3.8\,\mu$s, $r_\infty = 21.9$ MB/s, and $n_{\frac{1}{2}} = 53$ bytes, representing only modest increases in the startup latency and half-bandwidth packet size.

The `switch()` statement was added in the streaming receive loop to simulate the impact of even minimal packet interpretation. This change has little impact on overall latency, but because the overhead is added in the innermost loop, it is fully exposed for each packet and therefore has a much larger impact on bandwidth than other types of overhead. Note that the larger increase in latency from adding buffer management produces less reduction in bandwidth for short messages. Performance of the **hybrid + buffer management + switch()** is only $t_0 = 6.8\,\mu$s, $r_\infty = 21.8$ MB/s, and $n_{\frac{1}{2}} = 127$ bytes. While the peak bandwidth is nearly the same, there is a marked increase in $n_{\frac{1}{2}}$ by 74 bytes. Clearly, adding packet interpretation to the LCP would dramatically reduce short message performance.

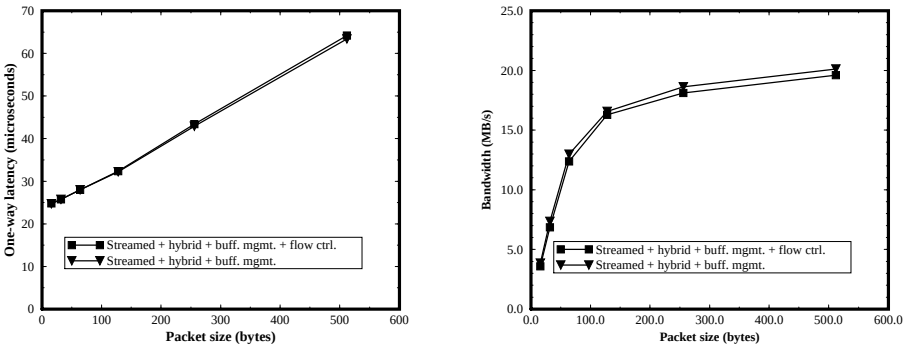
4.5 Flow Control

The final piece of a complete messaging layer is flow control. Because all networks have finite buffering, flow control is necessary to achieve reliable delivery, ensuring a receiver has enough buffer space to store incoming messages. Traditional flow control schemes include windows [28] which combine flow control and retransmission for fault tolerance. However window protocols generally require buffer space proportional to the number of senders, incurring large memory overheads in large clusters.

To avoid this overhead, FM implements a return-to-sender protocol which allocates buffers to prevent deadlock at the source (the reject queue), avoiding nonscalable buffering requirements. In return-to-sender, the sender optimistically sends packets into the network while reserving space locally for each outstanding packet. If the receiver does not have space, it rejects packets, retransmitting them to the sender. Successfully received packets are acknowledged, allowing source buffers to be released. Rejected packets are retransmitted eventually to ensure progress. Because each sender's buffering requirements are proportional to the number of outstanding packets, there is no large collection of buffers that must be statically allocated. Further, the buffer requirements for a particular node do not increase with the number of hosts in the system. The idea behind return-to-sender has been used in MPP's such as the Cray T3D, and is similar to deflection and chaos routing as used in the TERA-1 machine [2]. The well-known drawback of all of these retransmission schemes is

that delivery order is not preserved.

Because the reject queue holds packets rejected by **any** node, it can be though of as “network window” and provides an efficient use of pinned memory. Since the rejection mechanism does not provide fault-tolerance, the network is assumed to be reliable, or fault-tolerance must be provided by a higher level protocol. In the case of Myrinet, bit errors are exceedingly rare; other sources of system failure are much more likely. Multiple packets can be acknowledged with a single acknowledgement packet, and FM 1.0 optimizes further by piggybacking acknowledgements on ordinary data packets.



(a) Latency (b) Bandwidth

Figure 8: Fast Messages messaging layer performance

Comparing the messaging layers with and without flow control indicates that return-to-sender incurs little additional latency and only moderate loss in bandwidth (see Figure 8). The entire FM layer achieves $t_0 = 4.1\,\mu$s, $r_\infty = 21.4$ MB/s, and $n_{\frac{1}{2}} = 54$ bytes, a negligible difference from the performance of **streamed + hybrid + buffer management**.

4.6 Comparative Performance

There are few messaging layers available on the Myrinet today; the only major point of reference is the Myricom-supplied “Myrinet API.” This software is part of the standard Myrinet software distribution (version 2.0, available in March 1995) and used to support their TCP/IP implementations. Table 3 highlights some of the differences between the two messaging layers.

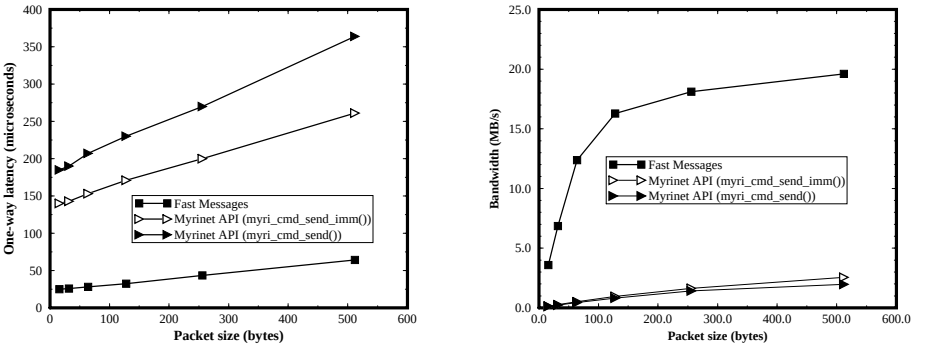
Feature	Fast Messages 1.0	Myrinet API 2.0
Data Movement	Direct from user space	From user space, DMA region, and supports scatter-gather operations
Delivery	Guaranteed	Not guaranteed
Delivery Order	No guarantee	Preserved
Reconfiguration	Manual	Automatic, continuous
Buffering	Large number of small buffers	Small number of large buffers
Fault Detection	Assumes reliable network	Message checksums

Table 3: Selected differences between Fast Messages and Myrinet API

From the preceding discussion, it should be clear that adding even the smallest feature to the LCP can exact a large penalty in performance. The Myricom API includes many additional features, each taking its toll on performance. For example, automatic network remapping—machines can be added or removed from the network without modifying any configuration files—may be convenient for users but can hurt the messaging layer's performance. Also, synchronization between the host and the LANai is expensive, yet must be done frequently in the Myrinet API, to pass buffer pointers back and forth.

The Myricom API's greater functionality and host-LANai synchronization structure translates to significantly poorer performance than FM's (see Figure 9). The Myricom API presents two interfaces, `myri_cmd_send_imm()` which uses the processor to move data to the LANai, and `myri_cmd_send()` which uses DMA. FM achieves superior performance to both interfaces. The design of FM achieves its goal of high performance for short messages reflected in low values for t_0 and $n_{\frac{1}{2}}$. Myricom's API has much greater basic latency and half power message size, but achieves comparable bandwidth at the peak. Specifically, for the Myricom API, $t_0 = 105.0\,\mu$s, $r_\infty = 23.9$ MB/s[3], and $n_{\frac{1}{2}} \geq 4,409$ bytes. For the modest sacrifice in peak bandwidth, we have achieved a reduction of $n_{\frac{1}{2}}$ of two orders of magnitude.

[3]The Myricom API does not support message sizes large enough to accurately measure r_∞, so we used the SBus write bandwidth.



(a) Latency

(b) Bandwidth

Figure 9: Fast Messages vs. Myricom's API

5 Discussion

Our design goal for FM was low latency and high bandwidth for short messages; as illustrated in Table 4, FM 1.0 achieves an $n_{\frac{1}{2}}$ of 54 bytes, delivering 10.7MB/s at this small packet size. Larger packet deliver higher bandwidth at some penalty in latency – 512 byte packets deliver 19.6 MB/s, greater than OC-3 ATM, and competitive with commercial massively-parallel machines. For example, while FM's latencies are larger than Active Messages on the CM-5, the bandwidth is much higher. FM also compares favorably to recent MPPs [20, 21] in both bandwidth and latency.

While there may appear to be many design tradeoffs involving performance for short or long messages (latency versus bandwidth), the design of FM is a counterexample. Despite consistently favoring low latency, the delivered peak bandwidth is within a few MB/s of the Myricom API. In fact, it may be most advantageous to pick frame sizes which deliver 80-90% of the achievable bandwidth; there is little bandwidth benefit in going beyond this size, and FM shows that low latencies are possible. Based on these considerations, we chose a 128-byte frame size for FM 1.0. Larger messages will require segmentation and reassembly into frames of this size. Our approach differs from the Generic Active Messages model [1] which provides extremely short (4 word) messages and longer network DMA transfers. Serendipitously, the FM frame size is close to the best size for supporting TCP/IP and UDP/IP traffic, where the vast majority of packets would fit into a single frame [5]. This presents the possibil-

Messaging layer feature					Performance metric		
Stream	SBus	Buffer	Flow	switch()	t_0 (μs)	r_∞ (MB/s)	$n_{\frac{1}{2}}$ (bytes)
	None				4.2	76.3	315
$\surd$	None				3.5	76.3	249
$\surd$	Hybrid				3.5	21.2	44
$\surd$	Hybrid	$\surd$			3.8	21.9	53
$\surd$	Hybrid	$\surd$	$\surd$		**4.1**	**21.4**	**54**
$\surd$	Hybrid	$\surd$		$\surd$	6.8	21.8	127
$\surd$	Hybrid	$\surd$	$\surd$	$\surd$	6.9	21.7	127
$\surd$	All DMA				7.5	33.0	162
Myrinet API (`myri_cmd_send_imm()`)					105	*23.9*	$\geq$ 4,4K
Myrinet API (`myri_cmd_send()`)					121	*23.9*	$\geq$ 6,9K

Table 4: Summary of FM 1.0 performance data

ity that a single low-level messaging layer can support both efficient parallel computation and traditional protocols.

As mentioned in Section 4.5, return-to-sender is an optimistic flow control protocol. Its potentially high performance is based on the assumption that the receiver polls the network in a timely manner, removing packets before its receive queue fills. However, the current implementation of return-to-sender implements rejection at the host (the LANai was too slow), which eliminates the memory requirement benefits. Interesting areas for future study include comparing return-to-sender to traditional window protocols, and exploring other dynamic flow control schemes.

FM's performance is a product of a design carefully optimized for low latency, subject to the constraints of a particular workstation and network interface architecture. However, we believe these design tradeoffs not only apply to many systems today, they are likely to apply to cluster systems in the future. Because of the advantages of input/output bus network interfaces, the basic network interface architecture addressed by the design of FM is likely to persist in great quantity. Further, the major factors that drove the FM design include relative speed of host and coprocessor, performance of input/output bus, memory capacity of coprocessor, and DMA restrictions. These characteristics have every indication of continuing in workstation cluster systems of the future.

Other researchers have built messaging layers for workstation clusters, using other commercial hardware [30, 25]. FM on Myrinet has performance beyond von Eicken et al.'s SPARCstation Active

Messages (SSAM), which employs ATM interface cards on the SBus. SSAM achieves 26 μs latency on 4-word messages, assuming a 10 μs switch latency. Our measurements through an 8-port Myricom switch achieve latencies of 25 μs for 4-word messages and 32 μs for 32-word messages. SSAM peaks at 7.5 MB/s, while FM achieves 16.2 MB/s for 32-word messages. Martin's HP Active Messages (HPAM) uses HP workstations with Medusa FDDI interface cards on a high speed graphics bus. This makes HPAM's network interface much closer to the processor and therefore, connected with much higher bandwidth. Consequently, HPAM achieves a lower latency, 15 μs for 4-word messages and peak bandwidth of 12 MB/s. Despite being hindered by an I/O bus network interface, FM delivers higher bandwidth. HPAM's hardware has the further advantage of significant memory (one megabyte versus 128 kilobytes for Myrinet) on the interface card. This is a key difference which affects the buffering protocols feasible in the two systems.

A number of other researchers have explored the development of special hardware to achieve low latency/high bandwidth communication (MINI [16], FUNet [18], VUNet [19], etc.). However, these hardware approaches have the drawback that they depend on specific memory bus interfaces, and require significant hardware investment. FM demonstrates that decent performance can be achieved without moving the interfaces closer to the host processor.

We believe that only modest architectural improvements are required to reduce the penalty of I/O bus interfaces further. The most useful improvement would be to improve workstation performance on SBus operations. Simply supporting burst-mode operations in the write buffer across the MBus-SBus interface would provide DMA-like bandwidth into the network enabling FM to achieve performance close to that of **streamed** shown in Figure 3. In addition, accelerating the LANai processor would reduce the serial overhead, a significant contributor to messaging latency. Building custom hardware that implements the functionality of FM's LCP is another means to reduce that serial overhead. Such custom hardware would provide truly concurrent service for sends and receives for both the host and network channel.

6 Conclusion

Illinois Fast Messages 1.0 is a high performance messaging layer that achieves communication performance comparable to that of an MPP on a workstation cluster connected with a Myrinet network. Because workstations are not designed to deliver low latency communication, we devised efficient solutions to a series of critical issues: division

of labor between host and network coprocessor, efficient utilization of the I/O bus, and implementation of scalable and efficient schemes for flow control and buffer management. FM 1.0 demonstrates that it is possible to find solutions in the context of current-day workstation and network interface architectures which deliver high performance. Despite our progress, we point out two minor changes that would have a significant impact on achievable network performance – improved I/O bus performance for non-DMA operations and a moderately faster network interface processor.

7 Future work

FM 1.0 provides the starting point for a wealth of research in making workstation clusters useful for parallel computation. There are significant outstanding questions about how to do flow control and reliable transmission efficiently. We are exploring the software and hardware issues in extending FM to provide higher performance, multitasking (protection), and preemptive messaging. In addition, delivering effective low-latency communication requires coordinated scheduling, so we are exploring integrating messaging with the node scheduler.

FM is designed to support efficient implementation of a variety of communication libraries and run-time systems. To explore interface, buffering, and scheduling issues, we are building implementations of MPI [14], TCP/IP [10], and the Illinois Concert system's runtime [8]. MPI is of growing popularity among application builders, presents interesting collective communication operations, and there are efficient implementations to compare against [15]. TCP/IP is a legacy protocol in widespread use. And the Illinois Concert system is a fine-grained programming system which depends critically on low-cost high performance communication.

References

[1] The Generic Active Message Interface Specification. Available from http://now.cs.berkeley.edu/Papers/Papers/gam_spec.ps, 1994.

[2] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. **1990 International Conf. on Supercomputing**, June 11-15 1990. Published as Computer Architecture News 18:3.

[3] T. Anderson, D. Culler, and D. Patterson. A case for NOW (networks of workstations). **IEEE Micro**, 15(1):54–64, 1995.

[4] T. M. Anderson and R. S. Cornelius. High-performance switching with Fibre Channel. In **Digest of Papers Compcon 1992**, pages 261–268. IEEE Computer Society Press, 1992. Los Alamitos, Calif.

[5] G. Armitage and K. Adams. How inefficient is IP over ATM anyway? **IEEE Network**, Jan/Feb 1995.

[6] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet—a gigabit-per-second local-area network. **IEEE Micro**, 15(1):29–36, February 1995. Available from http://www.myri.com/myricom/Hot.ps .

[7] CCITT, SG XVIII, Report R34. Draft Recommendation I.150: B-ISDN ATM functional characteristics, June 1990.

[8] Andrew A. Chien, Vijay Karamcheti, John Plevyak, and Xingbin Zhang. Concurrent aggregates language report 2.0. Available via anonymous ftp from cs.uiuc.edu in /pub/csag or from http://www-csag.cs.uiuc.edu/, September 1993.

[9] D. Clark, V. Jacobson, J Romkey, and H. Salwen. An analysis of TCP processing overhead. **IEEE Communication Magazine**, 27(6):23–29, June 1989.

[10] Douglas E. Comer. Internetworking with TCP/IP Vol I: Principles Protocols, and Architecture, 2nd edition. Prentice Hall, Englewood Cliffs, NJ, 1991.

[11] Cray Research, Inc. Cray T3D System Architecture Overview, March 1993.

[12] Peter Druschel and Larry L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In **Proceedings of Fourteenth ACM Symposium on Operating Systems Principles**, pages 189–202. ACM SIGOPS, ACM Press, December 1993.

[13] Fiber-distributed data interface (FDDI)—Token ring media access control (MAC). American National Standard for Information Systems ANSI X3.139-1987, July 1987. American National Standards Institute.

[14] Message Passing Interface Forum. The MPI message passing interface standard. Technical report, University of Tennessee, Knoxville, April 1994. Can be found at http://www.mcs.anl.gov/mpi/mpi-report.ps.

[15] H. Franke, C. E. Wu, M Riviere, P Pattnik, and M Snir. MPI programming environment for IBM SP1/SP2. In **Proceedings of the International Symposium on Computer Architecture**, 1995.

[16] F. Hady, R. Minnich, and D. Burns. The Memory Integrated Network Interface. In **Proceedings of the IEEE Symposium on Hot Interconnects**, 1994.

[17] Mark Henderson, Bill Nickless, and Rick Stevens. A scalable high-performance I/O system. In **Proceedings of the Scalable High-Performance Computing Conference**, pages 79–86, 1994.

[18] James Hoe and A. Boughton. Network substrate for parallel processing on a workstation cluster. In **Proceedings of the IEEE Symposium on Hot Interconnects**, 1994.

[19] H. Houh, J. Adam, M. Ismert, C. Lindblad, and D. Tennenhouse. The VuNet desk area network: Architecture, implementation and experience. **IEEE Journal of Selected Areas in Communications**, 1995.

[20] IBM 9076 Scalable POWERparallel 1: General information. IBM brochure GH26-7219-00, February 1993. Available from http://ibm.tc.cornell.edu/ibm/pps/sp2/index.html .

[21] Intel Corporation. Paragon XP/S Product Overview, 1991.

[22] Vijay Karamcheti and Andrew A. Chien. A comparison of architectural support for messaging on the TMC CM-5 and the Cray T3D. In **Proceedings of the International Symposium on Computer Architecture**, 1995. Available from http://www-csag.cs.uiuc.edu/papers/cm5-t3d-messaging.ps .

[23] Vijay Karamcheti and Andrew A. Chien. FM—fast messaging on the Cray T3D. Available from http://www-csag.cs.uiuc.edu/papers/t3d-fm-manual.ps, February 1995.

[24] M. Liu, J. Hsieh, D. Hu, J. Thomas, and J. MacDonald. Distributed network computing over Local ATM Networks. In **Supercomputing '94**, 1995.

[25] R. Martin. HPAM: An Active Message layer for a network of HP workstation. In **Proceedings of the IEEE Symposium on Hot Interconnects**, 1994. Available from ftp://ftp.cs.berkeley.edu/ucb/CASTLE/Active_Messages/hotipaper.ps.

[26] Meiko World Incorporated. Meiko Computing Surface Communications Processor Overview, 1993.

[27] V. S. Sunderam. PVM: A framework for parallel distributed computing. **Concurrency, Practice and Experience**, 2(4):315–340, [12] 1990.

[28] A. S. Tanenbaum. Computer networks. Prentice-Hall 2nd ed. 1989, 1981.

[29] Thinking Machines Corporation, 245 First Street, Cambridge, MA 02154-1264. The Connection Machine CM-5 Technical Summary, October 1991.

[30] T. von Eicken, A. Basu, and V. Buch. Low-latency communication over ATM networks using Active Messages. **IEEE Micro**, 15(1):46–53, 1995.

[31] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: a mechanism for integrated communication and computation. In **Proceedings of the International Symposium on Computer Architecture**, 1992. Available from http://www.cs.cornell.edu/Info/People/tve/ucb_papers/isca92.ps .

A Theoretical Peak Performance of LANai

Theoretical peak performance in Figures Figure 3 (a) and (b) is based on the following measured performance characteristics of the LANai.

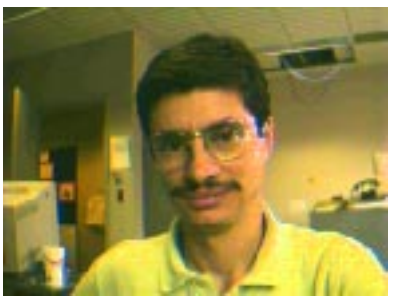
$$
\begin{aligned}
\text{DMA setup } (t_{\text{DMA}}) &= 8 \text{ cycles} \times 40 \frac{\text{ns}}{\text{cycle}} = 320 \text{ ns} \\
\text{Message overhead } (t_0) &= t_{\text{DMA}} + N \text{ bytes} \times 12.5 \frac{\text{ns}}{\text{byte}} = \\
&= (320 + 12.5N) \text{ ns} \\
\text{Message latency } (\ell) &= t_0 + t_{\text{switch}} = (320 + 12.5N) + \\
&\quad +550 = (870 + 12.5N) \text{ ns} \\
\text{Communication bandwidth } (r_N) &= \frac{\text{bytes}}{t_0} = \frac{N \text{ bytes}}{(320 + 12.5N) \text{ ns}}
\end{aligned}
$$

Scott Pakin

Scott Pakin is currently a Ph.D. student in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He has been working in the Concurrent Systems Architecture Group under Professor Andrew A. Chien since January, 1993. The primary goals of Scott's research involve architecting scalable, parallel systems, leveraging off commodity hardware, but developing novel software technology. Scott received his B.S. in mathematics/computer science from Carnegie Mellon University in 1992 and his M.S. in computer science from the University of Illinois at Urbana-Champaign in 1995.

Contact Information

E-mail:	pakin@cs.uiuc.edu
Telephone:	(217) 244-7116
Fax:	(217) 244-6500
Mailing address:	Scott Pakin Department of Computer Science 1304 W. Springfield Ave. Urbana, Illinois 61801 USA
www:	http://www-csag.cs.uiuc.edu/individual/pakin



Mario Lauria

Mario Lauria graduated in electronic engineering at the University of Naples, Italy, in 1992. He spent nine months at Ansaldo Trasporti, where he worked as a computer systems analyst. In 1994 he joined

the Department of Computer Science and Systems of the University of Naples, where he is working toward a Ph.D. in computer science. He is spending a Fulbright scholarship he was granted in 1994 at the University of Illinois at Urbana-Champaign, where he has joined the Concurrent Systems Architecture Group. His research interests include high performance computer communications and distributed simulation, with the realization of a high speed communication system for network of workstations as his present goal.

Contact Information

E-mail:	lauria@cs.uiuc.edu	lauria@cps.na.cnr.it
Phone:	(217) 244-7118	+39 81 768-2897
Mail:	Department of Computer Science 1304 W. Springfield Ave. Urbana, Illinois 61801 USA	Dipartimento di Informatica e Sistemistica via Claudio 21 80125 Napoli Italy
www:	http://www-csag.cs.uiuc.edu/individual/lauria	

Andrew A. Chien

Andrew A. Chien is currently an Associate Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign, where he holds a joint appointment as an Associate Professor in the Department of Electrical and Computer Engineering as well as a Research Scientist with the National Center for Supercomputing Applications (NCSA). The primary goals of Professor Chien's research involve the interaction of programming languages, compilers, system software, and machine architecture in high-performance parallel systems. He has participated in the design of a number of parallel systems (hardware and software), including the Illinois Concert System (efficient parallel object-oriented programming), and Illinois Fast Messages (high performance communication for MPP's and workstation clusters), and the MIT J-Machine (a 4096-processor fine-grained parallel computer). His research is supported by ARPA, NASA, ONR, and NSF as well as several corporate donors. Professor

Andrew Chien also co-directs an ARPA-funded I/O characterization project and is a participant in the Scalable I/O Initiative (SIO), working on the characterization of application input/output patterns for scalable, parallel scientific programs. Andrew Chien is the leader of the Concurrent Systems Architecture Group at the University of Illinois Dr. Chien received his B.S. in electrical engineering from the Massachusetts Institute of Technology in 1984 and his M.S. and Ph.D., in computer science, from the Massachusetts Institute of Technology in 1987 and 1990, respectively. He was a recipient of the 1994 National Science Foundation Young Investigator Award, and in 1995 received the C. W. Gear Outstanding Junior Faculty Award.

Contact Information

E-mail:	achien@cs.uiuc.edu
Telephone:	(217) 333-6844
Fax:	(217) 244-6500
Mailing address:	Andrew A. Chien Department of Computer Science 1304 W. Springfield Ave. Urbana, Illinois 61801 USA
www:	http://www-csag.cs.uiuc.edu/individual/achien