CHAPTER V

HOST-TO-HOST TRANSFERS

Modern message layers for cluster computers are optimized to provide high-bandwidth, low-latency communication between host CPUs in the cluster. While the key design goal of GRIM is flexible communication among host CPUs and peripheral devices, it is also desirable if GRIM is capable of providing reasonable levels of performance for host-to-host interactions. In order to achieve such performance GRIM had to be designed with communication mechanisms that operate in a streamlined manner. The active message and remote memory programming interfaces used in GRIM both rely on the same mechanisms for transferring data between a pair of hosts. These mechanisms use a communication pipeline that is comprised of three sets of data transfers: host-to-host, NI-to-NI, and NI-to-host. By optimizing each of these transfers it is possible to improve the overall performance of GRIM. This performance is further enhanced by end-to-end optimizations that increase the amount of overlap that takes place between the stages in the communication pipeline.

This chapter examines the low-level performance characteristics of GRIM for data transfers between host CPUs. This analysis takes place in two parts. First, the three stages of data transfer in the communication pipeline are examined individually. For each pipeline stage, data transfer characteristics are reported as well as measurements of the amount of overhead that is required by GRIM to perform stage-specific operations. Second, GRIM's performance is examined in the context of end-to-end transfers. In this effort the effects of pipeline and cut-through optimizations are inspected. End-to-end performance measurements are reported for two sets of hosts and two types of Myrinet NI card. These results are compared to the performance values of other message layers, and reveal that GRIM provides competitive performance levels for interactions between host CPUs. These measurements also indicate that GRIM is designed in a manner that allows the overhead of its sophisticated functionality to be hidden from the critical path.

5.1 Overview of the Host-to-Host Communication Path

In traditional cluster computers the main goal of the communication library is to rapidly transfer data from one host CPU to another. Since host CPUs are the only resource available for processing an application in these clusters, existing communication libraries have largely been optimized for high-performance host-to-host interactions. In resource-rich clusters the fundamental goal of the communication library is flexible communication, as the cluster provides diverse resources to assist in the processing of an application. However, it is still important that a message layer for resource-rich cluster computers be able to obtain reasonable levels of performance for host-to-host interactions, as host CPUs are expected to provide significant contributions to application processing in these clusters. Therefore a key part of examining GRIM is evaluating the communication path it provides between two host CPUs.

GRIM offers two different programming interfaces that can be utilized for host-to-host communication: active message and remote memory operations. As Figure 5.1 illustrates both of these programming interfaces utilize the same communication path between hosts.

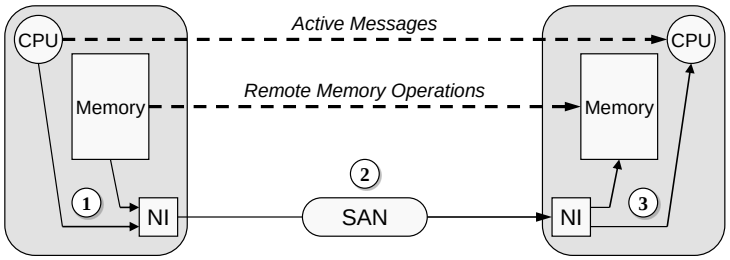

Figure 5.1: The active message and remote memory programming interfaces share the same communication path. The three phases of data transfer include (1) Host injection, (2) NI-NI delivery, and (3) NI ejection.

Data transfer in this path can be divided into three separate phases:

- **Host injection into the NI (Host-NI)**: Both the active message and remote memory interfaces begin the communication process by injecting a new message into the NI. This operation takes place over the local PCI bus with transfers orchestrated by the host CPU.
- **SAN Transfer (NI-NI)**: Data is then transferred across the SAN using reliable transmission mechanisms implemented in the NIs.
- **NI Ejection (NI-Host)**: After receiving a valid message the NI must transfer the message to the appropriate location. For active messages the NI appends an incoming message to the host's message queue. Remote memory operations are performed by the NI, where data is directly transferred to and from the host endpoint's address space.

Implementation details are provided in this chapter for each of the data transfer stages.

5.1.1 Evaluation Environment

Three different clusters were utilized in the performance benchmarks provided in this chapter. The first cluster utilizes hosts that have four 200-MHz Pentium Pro (PPro) processors and 32b/33MHz PCI. Due to the limited performance of these systems, the PPro hosts were only utilized to provide a point of comparison for PCI measurements. The second cluster used in this effort is based on hosts that have a single 550-MHz Pentium III (P3) processor and a 32b/33MHz PCI bus. These systems provide reasonable levels of performance and are characteristic of middle-of-the-road clusters that are commonly utilized in academic research efforts. The final cluster used in these tests employs hosts that contain dual 1.7-GHz Pentium IV (P4) processors and feature both 32b/33MHz and 64b/66MHz PCI buses. While the P4 hosts provide the best performance for all the clusters used in these tests, the motherboard chipset (Intel 860) for these hosts suffers from unusual PCI performance characteristics that are unsettling. Therefore, the majority of the benchmarks presented in this chapter were performed using the P3 cluster. Measurements using the P4 cluster are provided as part of the overall performance evaluation described in this chapter.

The measured values reported in this thesis are the results of benchmarking software that was constructed to provide practical and repeatable estimates of performance. In all

of these tests a measurement is performed several times. The median value for all iterations of a test is reported as the measured value. When applicable the benchmarking programs used in this work employ cache polluting mechanisms between successive iterations of a measurement. The mechanisms help to obtain a better measurement of performance under worst case operating conditions. Finally, it is important to note that benchmarks are performed on unloaded systems in order to obtain fair evaluation environment.

A common form of benchmarking network performance that is used in this chapter is to acquire a round-trip timing measurement for a transmission. In this form of measurement a message is transmitted from one entity to another and then returned to the original sender. The sender measures the amount of time the message is in-flight in the network and records this value as the round-trip time. Dividing the round-trip time in half yields an estimate of the one-way transmission time for the message. One-way transmission times are commonly referred to as the latency for communication. Dividing the amount of application data in a message by the one-way communication time provides an estimate of the bandwidth for the transmission.

5.2 Injecting Data into the Sending NI (Host-NI)

The first stage in the host-to-host communication path is for the host endpoint to inject a message into the local NI. This task is the same for active messages and remote memory operations because the sending host endpoint assembles an outgoing message in host memory and then transfers it to the NI's memory. There is no simple means of efficiently transferring data from a host-level application to a peripheral device such as the NI. Since the host CPU lacks a DMA engine of its own, it must either use the NI card's PCI DMA engine to perform the transfer or spend CPU cycles moving the data itself with programmed I/O (PIO) operations. While DMA operations rapidly transfer large blocks of data, configuring the DMA engines is a complex and an expensive operation. PIO mechanisms on the other hand are simple to implement but offer limited performance. Therefore researchers typically employ multiple mechanisms for injecting data from the host CPU into a NI and select the best mechanism for a transfer based on run-time conditions.

Since resource-rich clusters utilize a number of diverse peripheral devices, it is valuable to encapsulate the various host-to-card transfer mechanisms into a single portable library that can be used with different cluster resources. From a user's perspective it is beneficial if the library employs self-tuning mechanisms that allow the application to examine the host's hardware environment and automatically determine the most efficient means of injecting data into a peripheral device. Such a library has been constructed for GRIM named TPIL: the tunable PCI injection library. This library selects from multiple CPU-specific PIO and card-specific DMA transfer mechanisms to maximize the performance obtained for a given injection size. While this section specifically focuses on the use of TPIL to increase host-to-NI injection times, the library is utilized with other cards such as the Celoxica card discussed in the following chapter.

5.2.1 Programmed I/O Transfer Mechanisms

The first method by which data can be transferred from a host application to a peripheral device is through programmed I/O (PIO) operations. In this approach the device driver for a peripheral device provides an application with a memory map of the peripheral device's on-card memory. This memory appears as virtual memory that the application can read

or write. Interactions with the memory are translated by the host CPU and I/O chipsets into individual PCI read or write transactions. PIO operations incur a large amount of overhead (roughly 1 us for reads, 2 us for writes). In addition to providing a simple means of interacting with a device, PIO operations allow the host CPU to perform virtual memory translations for the data residing in host memory automatically through normal virtual memory operations.

A number of features of the modern x86 architecture can be exploited to accelerate PIO performance [49]. These hardware features include:

- **Write-Combining**: The write-combining MTRR registers included in Pentium MMX and higher processors allow stores to user-specified memory ranges to take place without strict ordering. This allows multiple writes to consecutive memory addresses to be combined for burst transfers.
- **MMX Registers**: The eight 64-bit MMX registers can be used as a temporary buffer for moving 64-byte blocks of data between host memory and the I/O system. This technique allows data writes to take place as burst operations that are efficiently mapped by the chipset into PCI transactions.
- **SSE Cache Control**: The streaming SIMD extensions (SSE) [90] unit adds features to provide user-level control of the CPU's cache. In addition to pre-fetching operations, the SSE hardware provides non-temporal stores where writes can bypass cache memory and be flushed directly to memory.

Previous literature [15] has discussed the use of write-combining to improve the host-to-card performance for transfers less than a kilobyte in size. While this greatly reduces the amount of time an application spends injecting data, there are pitfalls that must be addressed. The main hazard with write-combining is that writes can be reordered in the chipset to improve burst transfer performance. For NIs this could result in a race condition where an update to a queue pointer erroneously bypasses the actual placement of data in the queue. Such hazards must be prevented through careful definitions of memory regions that perform write-combining or by using fence instructions made available in Pentium III processors. A second pitfall is that there are a limited number of regions that can be marked for write-combining, and that the definition of such operations is a privileged operation. While early versions of GRIM utilized write-combining, it has been abandoned in favor of the MMX and SSE PIO transfer techniques.

5.2.2 DMA Transfer Mechanisms

The second means of injecting data into a peripheral device is to utilize DMA transfers. In this approach the host CPU configures a peripheral device's DMA engines to transfer data from host memory to card memory. DMA transfers are generally only useful for moving large blocks of data because there is a large amount of overhead involved in having the host orchestrate a DMA transaction. Part of this overhead is due to the fact that the host CPU must use PIO writes to configure the registers of a card's DMA engines for each transfer. Additional overhead can be attributed to the notification mechanisms employed by the DMA engines. After the host initiates a DMA transaction it must wait until the DMA completes before it can proceed. A DMA engine typically notifies the host that a transfer has completed by generating an interrupt, which must be handled by the card's device driver.

The main difficulty in utilizing DMA transfers is dealing with virtual memory. In the x86 architecture PCI devices operate with physical addresses while applications utilize virtual addresses. As a consequence three issues must be addressed when using a DMA operation: (i) virtual to physical address translation must be performed, (ii) data for a single DMA must be contiguous in the physical address space, and (iii) memory must not be changed (i.e. paged out) during a DMA transfer. Based on these issues designers typically employ one of two approaches to using DMAs. The first approach is simply to allocate a large block of pinned contiguous memory which serves as an intermediate buffer for data transferred with a DMA. While this approach simplifies DMA transfers it incurs the overhead of a data copy from application memory to the intermediate buffer. Another approach is to instead pin the user's virtual memory and perform DMA operations on the individual page frames housing the data. This approach involves multiple DMA operations but generally provides the best performance. In the TPIL library three card-specific operations are provided for transferring memory with DMA engines:

- **One-Copy**: In this approach user data is copied into a large (128 KB) contiguous buffer. The card then issues a single DMA to move the data. The operation is repeated if application data exceeds the capacity of the transfer buffer.
- **Double-Buffered One-Copy**: Like the previous approach data is copied from user space to a contiguous transfer buffer in host memory. However, this approach splits the buffer in half and overlaps the transfer of data into the buffer with the DMA operation.
- **Zero-Copy**: This approach pins the pages holding user data and configures the DMA engines to transfer data directly from the user pages. While individual DMA transfers are limited to a page in size, this approach removes the need to copy data in host memory, thus greatly improving speed.

5.2.3 TPIL Host-to-NI Performance

TPIL is designed to operate with the GNU/Linux 2.4 operating system and is implemented as a combination of user, kernel, and device level software. The internal benchmarking functions of TPIL were utilized to examine the performance of two types of hosts using two different versions of the Myrinet NI card. In the first set of tests, the Myrinet cards were placed in a 550-MHz Pentium III system that only featured a 32b/33MHz PCI bus. The results of the benchmark are displayed in Figure 5.2. In these tests the PIO methods had the best performance for small to medium sized transfers (less than 10 KB) while large transfers were best served with zero-copy DMA operations. The MMX and SSE methods had similar performance levels until approximately 2 KB, at which point the SSE's cache manipulation operations began to positively affect performance. For the DMA operations the zero-copy method provided the highest levels of performance for this system, with a maximum transfer rate of approximately 119 MB/s observed for both NI cards. These transfer rates are less than the maximum 132 MB/s performance levels of the 32b/33MHz PCI bus because the data transfers are sourced from virtual memory that is non-contiguous in the physical memory address space.

In a second series of tests the benchmarks were repeated using the 1.7 GHz Pentium IV hosts. The results of these tests are presented in Figure 5.3 for the (a) LANai 4 and (b) LANai 9 NI cards. The LANai 4 card was placed in a 32b/33MHz PCI slot and obtained

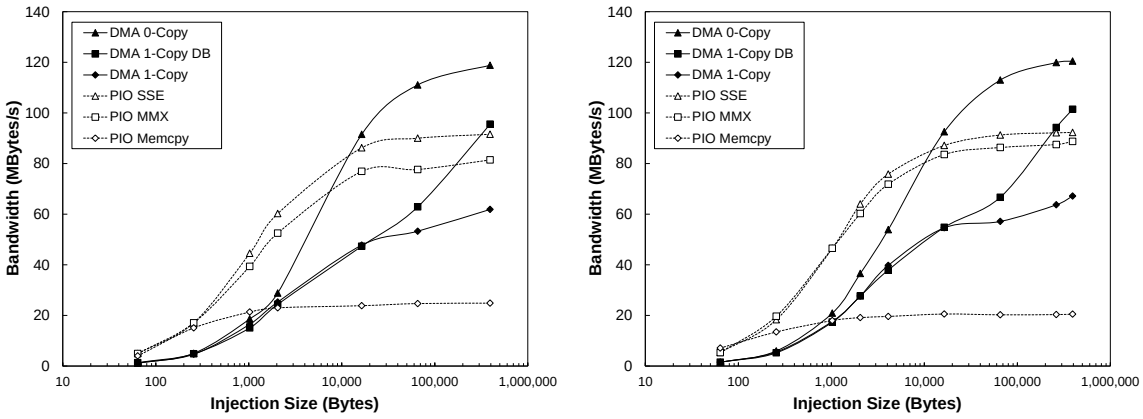

Figure 5.2: Host injection performance for a P3-550 MHz host using the (a) LANai 4 and (b) LANai 9 Myrinet NI cards.

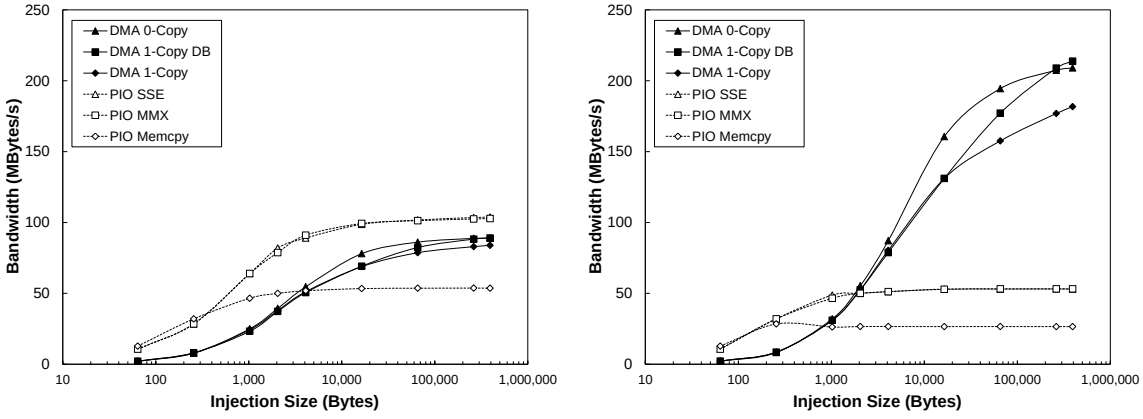

Figure 5.3: Host injection performance for a Pentium IV-1.7 GHz using (a) the LANai 4 (32b PCI) and (b) the LANai 9 (64b PCI) NI cards.

a maximum injection rate of 102 MB/s using SSE PIO transfers. The DMA engines for the LANai 4 card performed poorly in this host and allowed data injection rates of only 89 MB/s using zero-copy DMAs. The LANai 9 card was placed in a 64b/66MHz slot and obtained much better performance. While MMX and SSE PIO transfer mechanisms were limited to approximately 53 MB/s, the DMA operations were able to reach up to 213 MB/s. An interesting observation of this performance is that for very large transfers, the double-buffered one-copy DMA operation provides better performance than the zero-copy mechanism. This trait can be attributed to the fact that the Pentium IV hosts have large amounts of host-memory bandwidth because the hosts utilize RDRAM for main memory. Therefore for large transfers it is more efficient for the host to arrange source data so that DMAs take place in large contiguous transfers than it is for the host to schedule a large number of small transfers.

As a means of comparing PCI performance between systems, a LANai 9 card was placed in three different hosts and the injection performance of TPIL was measured. The best

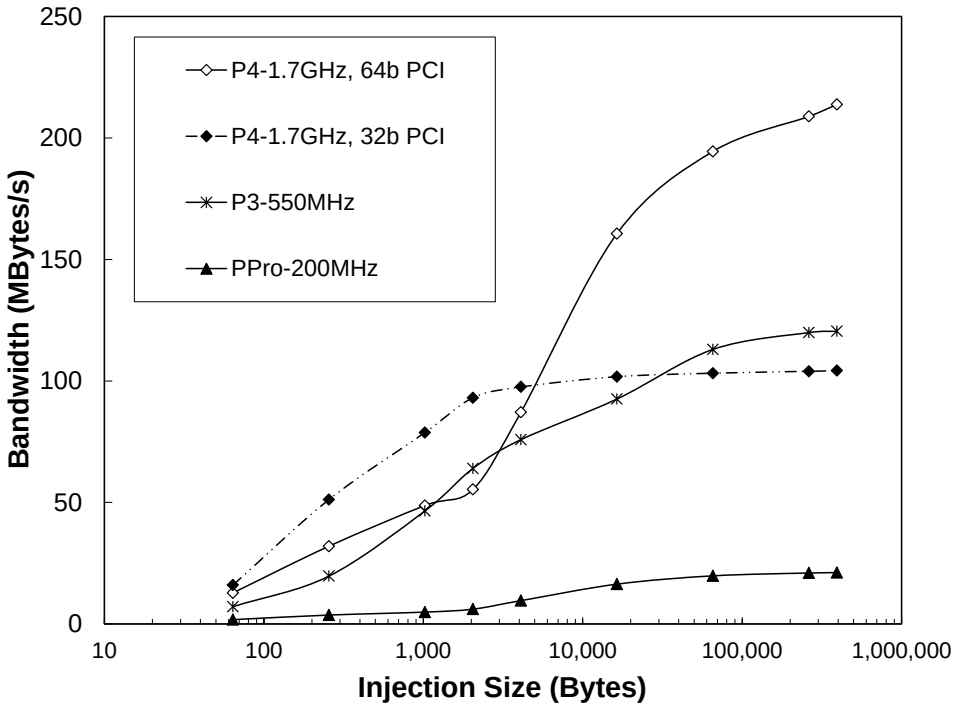

Figure 5.4: Overall TPIL performance for three different hosts.

transfer rates obtained with TPIL are reported in Figure 5.4 for each host. The PPro-200 MHz system provided the worst performance in these tests because it has poor PCI performance and lacks MMX and SSE units. The P3-550 MHz system provided the most linear performance of all the systems. Linear performance is desirable in a message layer because the user can be assured that reasonable performance can be obtained regardless of the size of the messages that are transferred. Finally the P4-1.7 GHz system exhibited different performance characteristics for its 32b and 64b PCI buses. PIO operations work well for the 32b PCI bus but not for the 64b PCI bus. The converse can be said of DMA operations for this system. Based on the performance measurements it is advantageous to either (a) place the LANai 9 card on the 32b PCI bus if the message layer does not issue injections larger than 4 KB or (b) construct the message layer to use transfer sizes that are greater than 8 KB when the LANai 9 card is placed on the 64b PCI bus. GRIM is optimized for the latter of these options because of the performance benefits of the 64b PCI bus.

5.3 Data Transfer between NI Pairs (NI-NI)

The second form of transmission in the host-to-host communication path is the transfer of data from the sending NI the receiving NI. This NI-NI transfer takes place across the Myrinet network and requires the use of a reliable transmission protocol in order to guarantee that messages are transferred in-order from one NI to another. As a first step in examining NI-NI performance, tests were constructed to observe the amount of time required to transfer various-sized messages between NI pairs. These measurements give an estimate of the native performance available in the SAN. Additional measurements were made of GRIM's NI firmware to determine how much overhead is added by GRIM's reliable

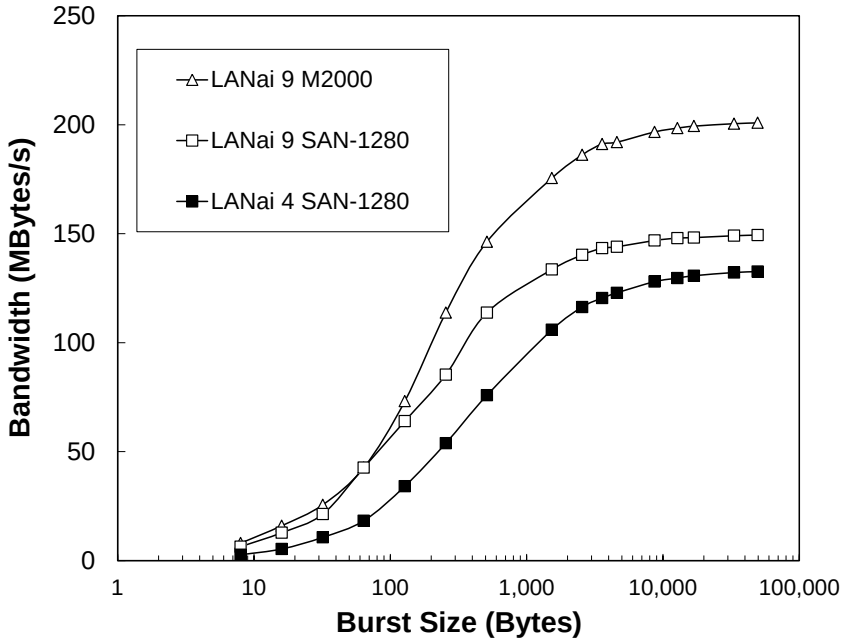

Figure 5.5: Observed bandwidth for different transfer sizes between NI pairs. Measurements are based on round-trip timings.

transmission protocols. These measurements include timings of the individual operations that are performed by sending and receiving NIs during the reliable transmission process.

5.3.1 Native SAN Performance

A first step in measuring the performance of NI-NI transfers in the host-to-host communication path is determining the native performance of the SAN hardware. A benchmarking program was constructed to determine how much bandwidth could be obtained from the SAN under ideal conditions. In this test round-trip timing measurements are performed between a pair of NIs that are directly connected by a SAN cable. NIs detect a new message in this test only when the message has arrived in its entirety at the NI. The test is performed several times using different values for message size.

The results of the bandwidths measured in this test are presented in Figure 5.5 for three pairs of NI cards. The first two tests measured the performance of the SAN-1280 links [3] using pairs of LANai 4 and LANai 9 NI cards. The LANai 4 cards are able to obtain approximately 132 MB/s (1.056 Gb/s) while the faster LANai 9 cards reach close to 150 MB/s (1.2 Gb/s). These transfer rates suggest that NI pairs can obtain a significant portion of the SAN-1280's available 160 MB/s (1.28 Gb/s) bandwidth. In the third test a two LANai 9 cards are configured to use the Myrinet-2000 link standard. These cards are able to obtain 200 MB/s (1.6 Gb/s), which is 80% of the Myrinet-2000 standard's 250 MB/s (2.0 Gb/s). It is important to note that the cards obtain good performance even for small messages. In each test half of the maximum observed bandwidth could be obtained using messages that were only 256 bytes long. This characteristic is especially important in cluster computing because many parallel applications frequently utilize small messages for state updates.

Table 5.1: Reliably delivering a message incurs overhead at the sending and receiving NIs.

NI	Function	Time (μs) LANai4	Time (μs) LANai9
Sending NI	Detect new message (1 Logical Channel / 8 Logical Channels)	1.0 / 5.5	0.5 / 1.5
	Set values in message	2.5	1.0
	Insert message in scoreboard	3.0	1.0
Receiving NI	Decode and beging processing message	2.5	1.0
	Verify sequencing information	2.0	0.5
	Destination capacity check	2.0	0.5
	ACK/NACK generation	4.0	1.5

5.3.2 Overhead for the Sending and Receiving NIs

While the previous tests demonstrate that a significant portion of the available SAN bandwidth can be obtained for data transmissions, there are a number of operations that take place at the sending and receiving NIs that add to the overhead of NI-NI communication. Because these operations degrade communication performance, it is beneficial to determine and measure the individual tasks that must be performed in the NI-NI transmission process. Table 5.1 provides a listing of measurements made to perform various operations in both the sending and receiving NIs for the LANai 4 and LANai 9 NI processors.

As this table indicates, the sending and receiving NIs must perform a number of tasks before and after a message is transmitted. In the sending NI the message must first be detected by scanning the NI's outgoing logical channels. Once a message is detected the sending NI must mark certain values in the message such as its sequence number and a token value that can be used to track the message. The sending NI must then record information about the message in a scoreboard, which allows the NI to keep track of the messages when retransmissions are necessary. At the receiving NI, an incoming message is first detected by the NI polling the network DMA interface. After a message has been detected the receiving NI must decode the message's header to determine how to process the message. For data messages the NI must verify that the message is marked with the next expected sequence number for the logical channel. Messages passing this test are then examined to determine if the specified destination endpoint has enough buffer space to accept the message. For accepted messages the NI must generate an acknowledgement message that is transmitted to the sender after the data message is transferred to the endpoint.

Performing these actions sequentially adds greatly to the overall overhead of the communication process between two NIs. Therefore the NI firmware is designed to operate in a manner that allows some of these operations to overlap with DMA transactions. For example in the sending NI scoreboard updates take place after the NI begins transmitting the outgoing message to the network. Likewise in the receiving NI the firmware begins decoding and processing a message as soon as the first few bytes of the message's header begin to arrive. While it is complex to construct such concurrency in NI firmware, doing so shaves overhead off of the critical path in the communication library. The fact that GRIM provides competitive performance to other less sophisticated message layers indicates that the overhead of GRIM's increased functionality can be effectively hidden.

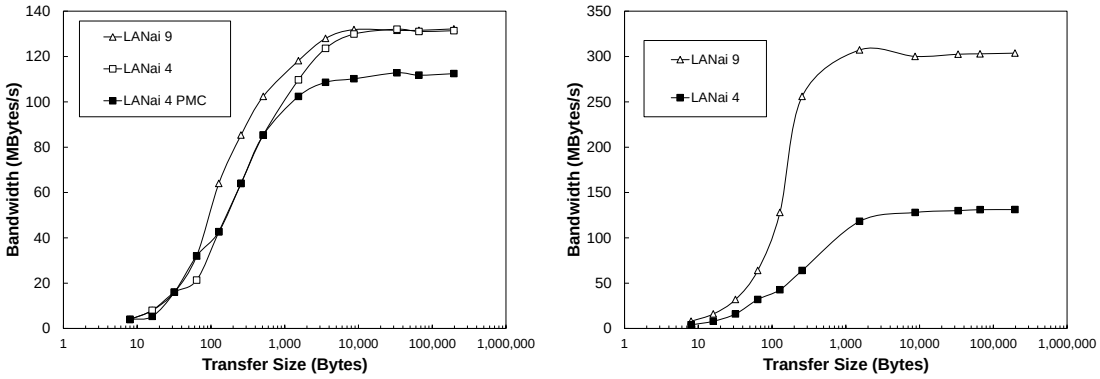

Figure 5.6: Bandwidth measurements for peripheral device DMA transfers into pinned host memory for (a) P3-550MHz and (b) P4-1.7GHz hosts.

5.4 Ejecting Data from the Receiving NI (NI-Host)

The last stage in the host-to-host communication path is the receiving NI's ejection of data to the destination endpoint. In this phase data from active messages and remote memory operations must be transferred to the proper locations in host memory. The NI accomplishes this task with the use of an on-card PCI DMA engine. For active messages data is appended to the back of a message queue for the host endpoint that is located in pinned, contiguous memory. A remote memory operation on the other hand requires the NI to transfer a block of data to host memory that is specified in the message. Completing this operation may require virtual to physical address translation by the NI depending on the arguments of the message.

5.4.1 Native NI PCI Performance

A first step in examining the performance of the NI-host ejection process is to determine the speed at which the NI can transfer data to host memory using its on-card DMA engines. A benchmark program was constructed in NI firmware to measure the amount of time required for a PCI device to transfer variable sized blocks of data from card memory to a contiguous region of host memory. This benchmark provides an estimate of the raw PCI performance a peripheral device can obtain from a host system. Three versions of the Myrinet NI card were used in this effort. The first two cards are the LANai 4 and LANai 9 NI cards that were used in the previous tests. The third card is a PCI mezzanine connector (PMC) version of the Myrinet LANai 4 card. This card is attached to one of the PMC daughter-card slots available on the Celoxica RC-1000 FPGA card that is discussed in Chapter 6. The RC-1000 card utilizes a PCI bridge unit to allow the LANai 4 PMC card to appear as a normal PCI device to the host system.

The benchmark software was utilized to measure the PCI performance of the P3-550 MHz and P4-1.7 GHz hosts. The results of the experiment are presented in Figure 5.6(a-b). For the P3-550 MHz host (a), the normal LANai 4 and LANai 9 cards were able to obtain up to 132 MB/s using the 32b PCI bus. The LANai 4 PMC card was only able to obtain approximately 112 MB/s. This degradation in performance can be attributed to the fact that the PMC card's PCI transactions are routed through the bridge chip of the RC-1000 card. For the P4-1.7 GHz host (b), the LANai 4 card obtained 130 MB/s from the 32b PCI

Table 5.2: The amount of time required for the receiving NI to deliver an active message to the host endpoint for a P3-550 MHz host.

Action	Time (μs)	
	LANai4	LANai9
DMA (32B / 64KB) message to host	3.0 / 496.0	2.0 / 494.0
Increment and convert queue pointer	1.5	0.5
DMA queue pointer to host	4.0	1.5

bus while the LANai 9 reached 307 MB/s from the 64b PCI bus. It is important to note that all of these cards were able to obtain reasonable performance levels, even with small transfer sizes. Half of the maximum observed bandwidth for these cards can be obtained using transfers that are only 256 bytes long.

5.4.2 Active Message Delivery

In the NI-host ejection stage in the host-to-host communication path, the NI implements separate delivery mechanisms for the active message and remote memory programming interfaces. The active message delivery mechanisms operate by inserting an incoming active message into a FIFO queue that is located in host memory. The NI is equipped with front and back indices for this queue so that it can determine where to place the next incoming message without assistance from the endpoint. Once a message has been transferred to the queue, the NI must notify the endpoint of the arrival of new data. In GRIM this notification is provided by the NI updating a queue pointer that is located in the endpoint's address space.

Instrumentation software was added to GRIM's NI firmware to determine how much time is required for the NI to perform its ejection tasks. Table 5.2 lists the amount of time the NI takes to perform the tasks for ejecting an active message. The first step in the ejection process is for the host to transfer the active message to the message to the proper location in the host's message queue. As the previous subsection discussed, it is possible for this task to be performed as data is arriving from the network in a cut-through fashion. The second task is for the NI to increment its local back pointer for the message queue. The new pointer is also converted into a value that has the same byte order as the destination endpoint (e.g., big-endian for the x86 host). The final task in ejecting an active message is to update the endpoint's back pointer using a DMA transfer. This transfer is only four bytes long and cannot be performed until the previous DMA completes.

5.4.3 Remote Memory Execution

For remote memory messages the NI must perform a DMA operation using arguments that are specified in the incoming message. Three different remote memory operations are possible. The first type of remote memory is a remote memory write to a physical memory address (RM-P). The NI processes RM-P messages by transferring the payload of the message to a physical address specified in the header of the message. As such these messages are executed immediately upon arrival at the NI without any external translation assistance. The remaining two remote memory operations utilize virtual addresses to reference

Table 5.3: The amount of time required for the receiving NI to process a remote memory operation in a P3-550 MHz host.

Action	Time (μs) LANai4	Time (μs) LANai9
Search NI VM translation cache (hit / miss)	3 / 10	1 / 4
DMA translation request to host	3	1.5
Host interrupt overhead	6.5	6
Host VM translation and NI update (1 page / 17 pages)	2 / 17	3 / 10
DMA message payload (4B / 64KB)	3 / 514	2 / 507
DMA update to lock variable (optional) (cached / non-cached address)	9 / 29	4 / 17

memory and therefore require the NI to perform address translation. The virtual memory write operation (RM-V) writes payload data to a virtual address while the virtual memory read operation (RM-RV) operation transmits the contents of a block of virtual memory to the sender. All remote memory operations provide an optional mechanism for updating a separate lock variable in virtual memory when a remote memory operation completes.

Instrumentation software was constructed to determine the amount of time required for the NI to process remote memory messages. Timings are reported in Table 5.3 for the LANai 4 and 9 NI cards. The first step in processing a remote memory operation is translating the virtual memory addresses supplied in the RM-V and RM-RV operations. For these operations the NI first consults a small cache of translations that is located in NI memory. If a translation cannot be obtained from this cache the NI must DMA a formal translation request to host memory and interrupt the host. The device driver for the NI parses these requests, translates the requested virtual memory addresses, and stores the results back into NI memory. It is important to note that translating virtual memory address is relatively expensive. However, translation overhead can partially be hidden by allowing translation to take place while the message's payload is arriving. After the NI is equipped with the proper physical address it can begin transferring data between the card and the host. Once this operation completes a remote memory operation can optionally update a lock value in host memory. This operation requires a virtual memory translation as well as a DMA of 4 bytes.

5.5 Performance and Optimizations of the Communication Path

While it is important to consider the performance of individual stages in the communication path, it is also necessary to consider how the stages behave in the context of end-to-end communication. One method of transmitting data through multiple network elements is to utilize a store-and-forward communication model. In this model each network element must receive a data message in its entirety before the message can be transmitted to the next stage. While this model has poor performance for individual transmissions, it is possible to use store-and-forward transmissions in a pipelined manner for improving the performance of

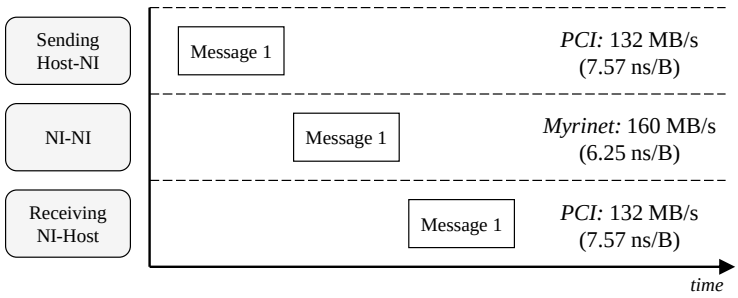

Figure 5.7: Messages are moved in their entirety in a store-and-forward communication model. The minimum amount of time required to transmit a byte can be determined by inverting the maximum bandwidth of the transmission mechanism

a series of transmissions. GRIM utilizes built-in fragmentation mechanisms to allow a store-and-forwards pipeline to be constructed in the communication library. These mechanisms allow data to be transferred in a high performance manner between host endpoints.

The performance of a store-and-forward system can be improved through the use of cut-through optimizations. In cut-through approaches network elements are permitted to begin transmitting a data message before the entire message has arrived at the network element. Cut-through techniques therefore reduce the amount of time between when a message begins to arrive at a network element and when transmission begins to the next stage in the communication pipeline. Cut-through optimizations have been applied in GRIM to both the sending and receiving NIs. While cut-through benefits are more visible at the receiving NI, both the sending and receiving NI cut-through optimizations provide noticeable improvements for the communication pipeline. This section provides details of the optimizations used to increase the performance of the host-to-host communication path, as well as performance measurements of each optimization.

5.5.1 Store-and-Forward Communication Model

One approach to implementing a system that delivers data messages through a multi-stage communication path is to employ store-and-forward data transfers. In this approach each stage in the communication path must receive a message in its entirety before it can begin transmitting the message to the next stage. An example of how store-and-forward data transfers take place in the host-to-host communication path is illustrated in Figure 5.7. In this example the host-NI, NI-NI, and NI-host transmissions of a data message take place sequentially with no overlap. Because a message must serially propagate through all three transmission stages, it should be expected that the performance of the overall communication path will be roughly a third of the performance of an individual transmission stage. Using the maximum bandwidth available for each transmission stage (listed to the right of the figure), it is possible to determine the maximum bandwidth that can be obtained for the transmission of a single message through the overall communication path. Inverting the maximum bandwidth for a transmission stage yields the amount of time required to transfer a single byte through a stage. Assuming 32b PCI buses, the sum of the transmission times for the three stages is 21.39 ns. This value corresponds to a maximum host-to-host bandwidth of 46.75 MB/s.

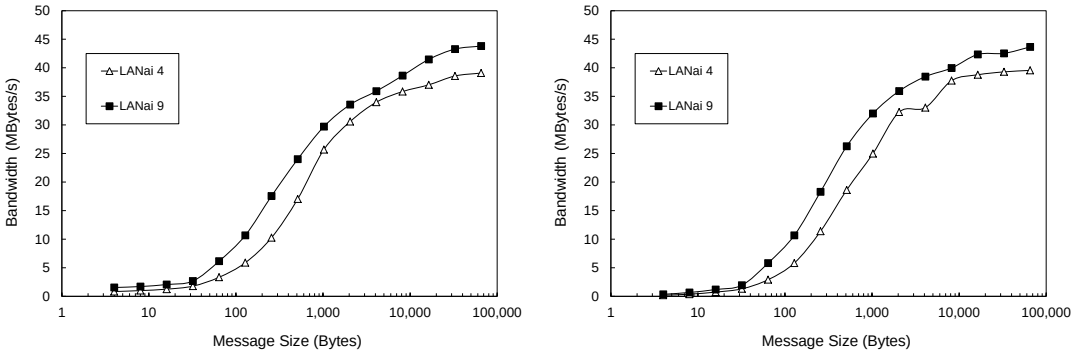

Figure 5.8: The store-and-forward performance for a single message transmission between a pair of P3-550MHz hosts using (a) active messages and (b) remote memory operations.

A set of benchmark programs were constructed to observe GRIM's host-to-host performance. These programs use round-trip timing measurements between two hosts to determine the overall bandwidth that can be obtained from the hosts. The tests are performed for both the active message and remote memory programming interfaces. In the active message tests a special function handler is used to either return an incoming message to the sender or stop a timer if the host is the endpoint that originally transmitted the message. In the remote memory tests a block of data is transferred using the remote memory write physical (RM-P) operation. The notification mechanisms of the RM-P operation are used to update a memory location that holds a value that signifies the completion of a transfer. The sending and receiving endpoints in the remote memory test poll this notification variable and transmit blocks of data when necessary. These benchmark programs are used in all of the tests in this chapter that examine host-to-host performance.

The host-to-host benchmarking programs were utilized to observe the performance of GRIM for the store-and-forward transmission of a single message between two P3-550 MHz hosts. The results of the experiments are presented in Figure 5.8(a-b) for message sizes ranging from four bytes to nearly 64 KB. For the tests using the active message programming interface (a), the LANai 4 and 9 NI cards reached maximum bandwidths of 39 MB/s and 43 MB/s respectively. The remote memory tests yielded nearly identical results. As expected the end-to-end performance of the system is less than the theoretical maximum bandwidth of 46.75 MB/s. This reduction in performance is due to processing overhead that takes place in the various stages of the communication path.

5.5.2 Store-and-Forward Pipelining

While the store-and-forward model of communication only offers limited performance for transferring a single message, it provides a framework for establishing a high-performance communication pipeline between a pair of endpoints. In a pipelined approach a series of messages is transmitted from one host to another in rapid secession. While each pipeline stage can only forward one message at a time, the abundance of messages to transfer allows each stage to operate at the same time on different messages. An example of how this concurrency can result in increased performance is illustrated in Figure 5.9. In this example a series of messages are transmitted from one host to another. After the host finishes injecting the first message to the sending NI, it can begin injecting the second

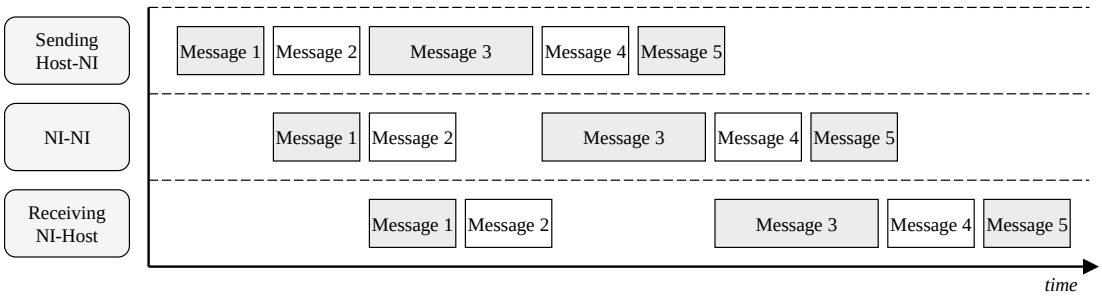

Figure 5.9: Store-and-forward mechanisms can be used to create a communication pipeline for increased performance.

message. During this time the sending NI can begin transmitting the first message to the receiving NI. As the pipeline fills it becomes possible for more stages to operate concurrently, increasing the performance of the overall communication path. It is important to note that pipeline performance is dependent on the sizes of the messages that are being transferred as well as the transfer rates of the individual pipeline stages. For example in Figure 5.9, the third message is larger than the second message. Therefore there is a gap between when the NI-NI stage finishes transmitting the second message and when it can begin transmitting the third message.

Given the performance advantages of pipelining it is beneficial to include mechanisms in the message layer that allow transmissions to take place in a pipelined fashion. One means of accomplishing this task is to utilize fragmentation and reassembly techniques at the programming interface level. In this approach large messages are broken into a series of smaller messages that are individually transmitted through the communication path and reassembled at the receiver. Because network hardware generally limits data transfers to a maximum transfer unit (MTU), most message layers naturally provide some form of fragmentation and reassembly. GRIM includes fragmentation and reassembly mechanisms for both the active message and remote memory programming interfaces. Low-level details of these mechanisms are provided in Chapter 8. These mechanisms were adapted to allow pipelining to take place in the communication path.

5.5.3 Pipelined Store-and-Forward Performance

The benchmarking programs used in the store-and-forward tests were modified to examine how fragment size affects the performance of the overall communication pipeline. In these tests the fragmentation size for the communication library was varied from 256 bytes to nearly 64 KB (GRIM's MTU). Host-to-host bandwidth was then measured using different sized messages for both the active message and the remote memory test programs. The tests were performed for P3-550 MHz hosts using a pair of LANai 4 NIs and a pair of LANai 9 NIs.

The results of the experiments are presented in Figure 5.10(a-d). The first observation to be made from these measurements is that pipelining does in fact increase communication performance for messages that are larger than approximately 4 KB. For these messages there is enough data being transferred that a message can be fragmented in a manner that allows concurrency between pipeline stages. As expected best results in these tests were obtained when the fragment size was selected in the middle range of possible values. For

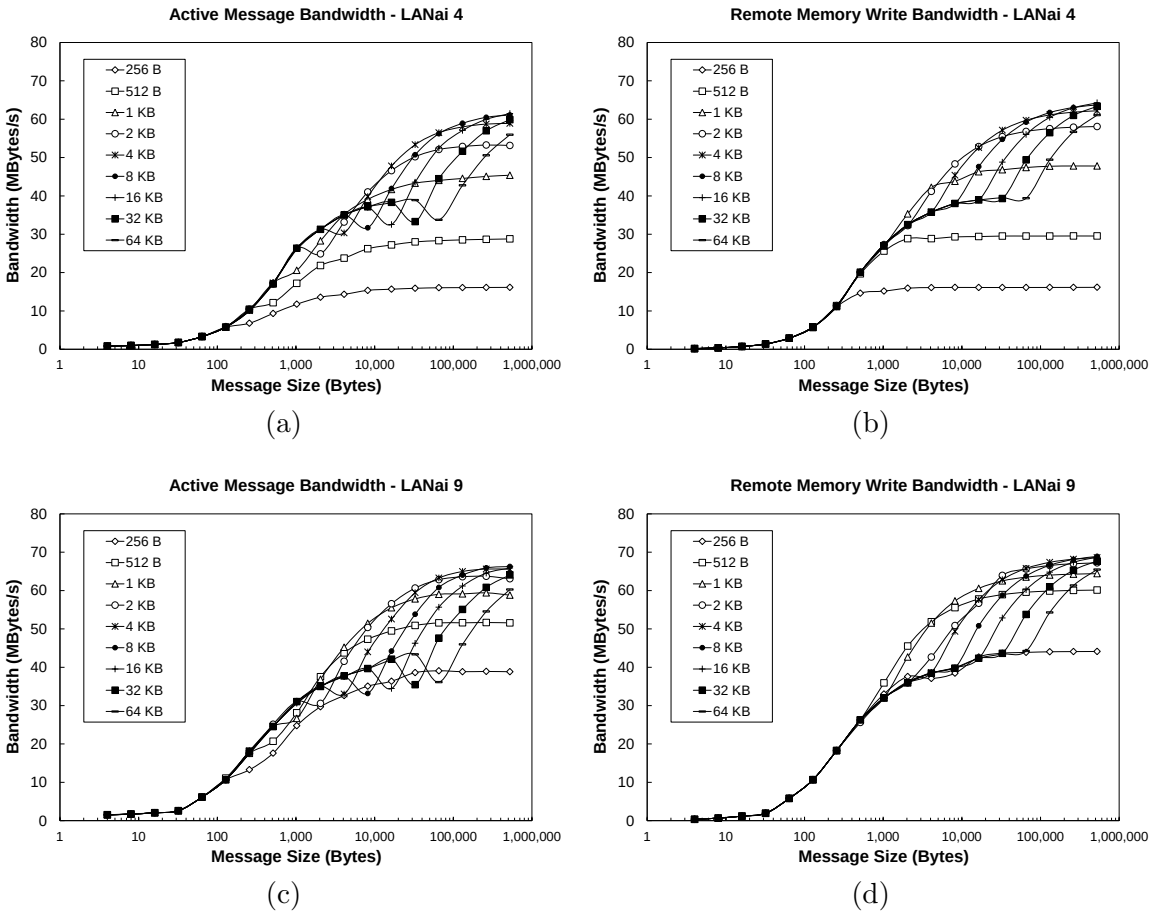

Figure 5.10: The performance of different fragment sizes for a pair of P3-550 MHz hosts using different NIs and different programming interfaces. The tests used (a) active messages with the LANai 4, (b) remote memory operations with the LANai 4, (c) active messages with the LANai 9, and (d) remote memory operations with the LANai 9.

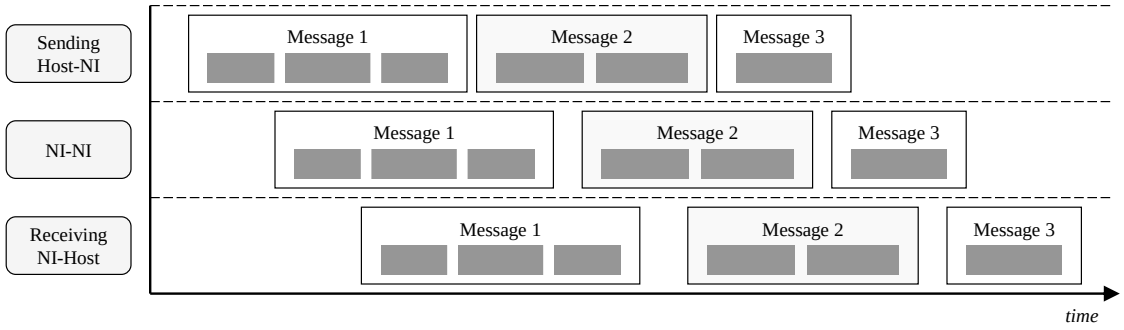

Figure 5.11: Cut-through optimizations allow a pipeline stage to begin transmitting a message before all of its data has arrived.

the LANai 4 cards a fragment size of approximately 2-4 KB provides good performance for both programming interfaces. The LANai 9 cards operate well with 1-2 KB fragments. The lower desirable fragment size can be attributed to the fact that the LANai 9 card is roughly three times faster than the LANai 4 card, thereby allowing a finer granularity of transmissions in the pipeline.

The benchmark tests also reveal that there are performance differences between the active message and remote memory interfaces. The most notable difference is that the remote memory performance curves generally increase with message size while the active message curves have slight performance drops at certain message sizes. These drops can be attributed to the reassembly mechanisms of the active message interface. Fragmented messages in the active message interface are reassembled in an intermediate message buffer. Therefore as soon as message fragmentation takes place there is a slight drop in performance because the receiver must perform an extra copy of all message fragments. The remote memory reassembly procedures do not need to perform this procedure and therefore function more efficiently.

5.5.4 Cut-through Optimizations

While pipelining increases the performance of the host-to-host communication path, a criticism is that data transfers are still based on store-and-forward mechanisms. These mechanisms can cause a pipeline stage to delay the transmission of a message until the stage has received the entire data message. An alternative approach is to employ cut-through routing, where individual stages are permitted to begin transmitting a message as soon as the first bytes of the message arrive. For many LAN cards cut-through optimizations are not possible because network interactions take place at the packet level. However, Myrinet NI cards allow users to manage network interactions at the byte level. Therefore it is possible to implement cut-through optimizations in Myrinet at both the sending and receiving NIs. An example of how cut-through optimization can be applied to the end-to-end communication path is illustrated in Figure 5.11. In this example a data message is transferred as a series of smaller segments. Each stage in the communication pipeline can therefore begin transmitting a message as soon as the first segment of a message arrives.

GRIM implements cut-through transfer optimizations for both the sending and receiving NIs. The receiving NI implementation is the more straightforward of the two because the

receiving NI can easily coordinate all necessary data transfers. In the current implementation the receiving NI monitors the capacity of an incoming message buffer and then begins transferring data to the host as soon as the network begins to fill the buffer. Implementing cut-through optimizations in the sending NI is more challenging because of the manner in which data injections are performed in GRIM. In other message layers [], the sending NI pulls a data message out of a host endpoint's address space and pushes the message to the network. As such it is trivial to implement the transfers in a cut-through manner because the transfers are performed entirely by the NI. Unfortunately, this approach is not valid for GRIM because messages are injected into the NI by endpoints in a push fashion. This push methodology is due to the fact that some endpoints in GRIM are peripheral devices that operate with simple transfer mechanisms.

Given the benefit of cut-through optimizations, special functions were constructed in GRIM to allow an injecting endpoint to achieve cut-through data transfers without having to resort to using the NI to pull messages into the NI. In GRIM the injecting endpoint and the sending NI can operate in a *cooperative cut-through* manner. In this effort the endpoint breaks the injection process of large messages into a series of smaller message segment injections. After transferring a segment to the NI, the endpoint updates a counter in the NI that specifies how much of the message has been transferred. When the sending NI detects a new message it begins transmitting as much of the message as is available to the wire. The NI then appends the network transmission as new segments arrive. This approach is cooperative in that in the common case, the endpoint and NI operate at the same time and transfer a message to the network in an efficient cut-through manner. However, there is no guarantee that the endpoint and sending NI will be synchronized to perform a cut-through transfer. In addition to increasing performance, this approach is advantageous because cut-through transfers can be accomplished without major modifications to the endpoint or NI software.

5.5.5 Performance with Cut-through Optimizations

The host-to-host performance benchmarks were used to examine the impact of cut-through optimizations on the communication pipeline. In these tests a fixed fragment size of 4 KB was selected for the transfers. Sending cut-though procedures were designed to segment a message into 1 KB blocks, while receiving cut-through mechanisms were designed to move data to the host endpoint as soon as it becomes available. The tests were performed for different message sizes using the LANai 4 and LANai 9 cards, with different cut-through optimizations enabled in each run. These tests utilized only the remote memory operation for the transfer, although active message performance provided similar behavior.

The results of the experiment are presented in Figure 5.12(a-b). Cut-through optimizations in these tests provided a significant performance boost for both the LANai 4 and 9 NI cards. Receiver-based cut-through provided the most improvement in the tests due to the fact that it can operate at a fine granularity. In the best case the receiver cut-through mechanisms can begin transferring data to the host as soon as the header for the message arrives. Sender-based cut-through provides a slight performance gain. This gain is less than the receiver-based optimizations because the 4 KB message fragment size limits the sender to four injection segments per fragment.

Additional tests were performed varying the GRIM's fragment and segment sizes. Determining a good combination of these settings is highly dependent on the PCI transfer characteristics of the host and NI cards. For both LANai 4 and 9 NI cards, the performance

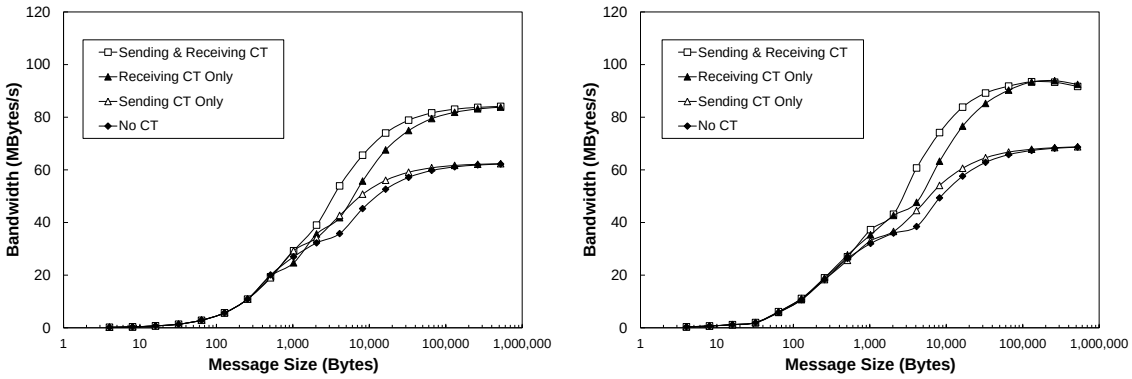

Figure 5.12: The effects of cut-through optimizations on end-to-end performance between P3-550MHz hosts using the (a) LANai 4 and (b) LANai 9 NI cards. These tests use RM-P programming interface, a fixed cut-through injection size of 1 KB, and a pipeline fragment size of 4 KB.

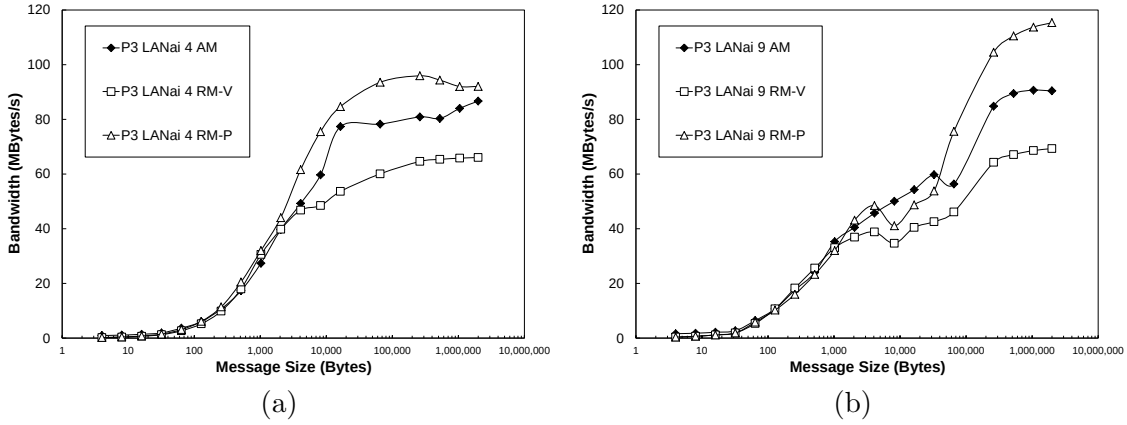

Figure 5.13: Overall performance of P3-550 MHz hosts in GRIM using (a) LANai 4 and (b) LANai 9 NI cards.

bottleneck is the transfer of data from host to NI card. The LANai 4 card obtains maximum PCI injection performance for 1-2 KB transfers. Therefore GRIM is configured with a segment size of 1 KB and a fragment size of 16 KB for the LANai 4 card. For the LANai 9 card, the DMA engines provide maximum performance at roughly 16 KB. Therefore GRIM is configured to use a segment size of 16 KB and a fragment size of 64 KB for the LANai 9 card.

5.6 Overall Performance

The host-to-host benchmark programs were run a final time with all performance optimizations enabled. The three programming interfaces were independently measured in this effort using message sizes ranging from four bytes to two megabytes. The LANai 4 and 9 NI cards were used to connect pairs of P3-550 MHz and P4-1.7 GHz hosts.

The results of the experiments with the P3-550 MHz hosts are presented in Figure 5.13(a-b). For the LANai 4 NI cards (a), the performance curves are relatively smooth. The RM-P

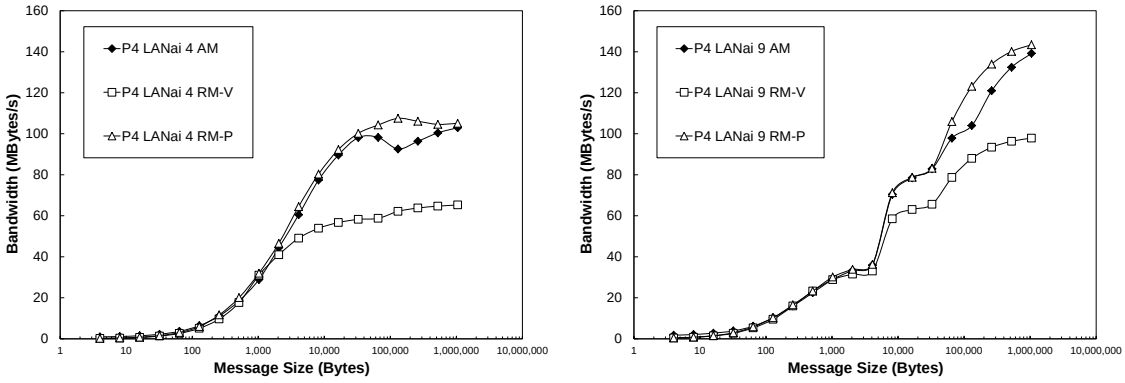

Figure 5.14: Overall performance of P4-1.7 GHz hosts in GRIM using (a) LANai 4 and (b) LANai 9 NI cards.

Table 5.4: The overall performance of GRIM for different transfer mechanisms on 550 MHz Pentium III systems (P3) and 1.7 GHz Pentium IV (P4) systems.

Host	PCI	NI	Latency (μs)			Bandwidth (MB/s)		
			AM	RM-V	RM-P	AM	RM-V	RM-P
P3	32b/33MHz	LANai 4	16	18	14	87	66	96
		LANai 9	10	10.5	9.5	102	69	116
P4	32b/33MHz	LANai 4	17	18.5	14.5	105	65	108
	64b/66MHz	LANai 9	9	8.5	8	144	98	146

interface provides the maximum performance of 96 MB/s and the minimum latency of 14 μs. The LANai 9 card (b) offers better performance in this host, reaching a maximum bandwidth of 116 MB/s (928 Mb/s) and a latency of 9.5 μs for RM-P operations. The transition from PIO to DMA injection mechanisms for this card results in performance reaching a temporary plateau for messages between 4 KB and 32 KB.

The results of the experiments for the P4-1.7 GHz hosts are presented in Figure 5.14(a-b). The LANai 4 cards (a) had to be placed in 32b PCI slots in the P4 hosts. These cards were able to obtain 108 MB/s of bandwidth and 14.5 μs of latency between the P4 hosts using the RM-P interface. Compared to the P3 tests, the P4s provide better bandwidth but slightly worse latency for the LANai 4 card. The P4's superior processing power also helps the more computationally demanding AM interface to provide performance that is closer to the RM-P interface than is observed in the P3 tests. LANai 9 cards (b) were placed in the 64b PCI slots of a pair of P4 hosts. In the performance tests the RM-P interface obtains 146 MB/s (1.168 Gb/s) of bandwidth and a minimum latency of 8 μs. Given that the SAN-1280 links offer a theoretical transfer rate of 160 MB/s, GRIM obtains a substantial portion of the available host-to-host performance.

GRIM's overall performance numbers are summarized in Table 5.4. An important observation of these numbers is that the RM-V interface provides only 60-75% of the performance of the RM-P interface. This performance degradation can be attributed to the overhead of translating virtual memory addresses at the receiving NI. The reason why this translation

has such a negative impact on RM-V performance is that the receiving NI must perform this operation before an incoming RM-V message can be processed. Because the translation is the first step in the NI's receiving process, it is difficult to hide the overhead of the operation with other transfers the NI must perform. Therefore users should be aware that the RM-V interface is only capable of providing limited performance.

5.7 Comparison with Other Message Layers

Over the years a number of message layers have been constructed for Myrinet. It is beneficial to compare the host-to-host performance of GRIM to these message layers in order to gauge how well GRIM can perform in traditional cluster applications. These comparisons also reveal how well the complexities of GRIM's lower mechanics are hidden from the critical path for end-to-end communication. Unfortunately many of the existing message layers are no longer supported and cannot be run on modern systems. As a means of comparison, this section provides two forms of performance estimates for existing message layers. First, performance estimates are provided for many message layers using the values reported by the original researchers. Second, performance estimates of the most commonly used message layer, GM, are provided for the same systems used in the GRIM benchmarks.

5.7.1 Reported Performance

It is important to compare the performance of GRIM with existing message layers for Myrinet. Performing such a comparison is challenging to do in an accurate and fair manner for a number of reasons. At a fundamental level GRIM is designed to provide functionality that is not present in other message layers. Therefore comparisons must be limited to traditional host-to-host metrics. Unfortunately, utilizing previously reported values in this comparison can often be misleading, as different utilize different hardware and software platforms, and sometimes different definitions of performance metrics. Ideally a fair comparison would run a standard set of benchmarks on the message layers using the same hardware and software environment. The hardship in this effort is that several of the message layers are now legacy software that is no longer supported due to changes in the Linux kernel or incompatibilities with modern hardware. Rather than port these legacy message layers to modern systems, the first part of comparing the performance of GRIM is to provide the performance measurements that were originally reported by the researchers.

Reported performance estimates for a number of Myrinet message layers are listed in Table 5.5. The majority of these measurements are based on older hosts using the LANai 4 NI card. Therefore the most relevant performance measurements of GRIM are those made of the P3-550MHz hosts that are equipped with the LANai 4 NI. In these tests GRIM obtained a maximum bandwidth of 96 MB/s and a minimum latency of 14 μs. In terms of latency GRIM offers slightly less performance than most of the message layers, but is still within an acceptable range. In terms of bandwidth GRIM is relatively competitive with other message layers.

5.7.2 Measured Performance

As a means of providing a more accurate comparison of GRIM's performance with other message layers, Myricom's GM [64] message layer was benchmarked for the P3 and P4 clusters. GM is the de facto standard for communication in Myrinet clusters and supports a variety of operating systems and NI cards. GM's internal benchmarking programs were

Table 5.5: Performance reported for various Myrinet message layers.

Message Layer	Host	NI	Latency (μs)	Bandwidth (MB/s)
AM [28]	UltraSparc 167 MHz (SBUS)	LANai 3	10	38
AM-II [23]	UltraSparc 167 MHz (SBUS)	LANai 4	21	31
BIP [75]	PPro-200 MHz	LANai 4	5	126
FM [70]	PPro-200 MHz	LANai 4	11	76.2
GM [64]	P3-1 GHz (64b PCI)	LANai 9c (Myrinet2000)	7	240
LFC [12]	PPro-200 MHz	LANai 4	12	65
Trapeze [101]	P3-450 MHz	LANai 4	30	110

Table 5.6: Measured performance for GM and GRIM on 550 MHz Pentium III (P3) systems and 1.7 GHz Pentium IV (P4) systems.

Host	PCI	NI	Latency (μs)		Bandwidth (MB/s)	
			GM	GRIM	GM	GRIM
P3	32b/33MHz	LANai 4	24	14	79	96
		LANai 9	9.7	9.5	108	116
P4	32b/33MHz	LANai 4	24	14.5	69	108
	64b/66MHz	LANai 9	9.44	8	146	146

utilized to determine how well the message layer performed using the same hardware that the GRIM benchmarks were performed with.

The results of the GM benchmarking experiments are presented in Table 5.6. In all of these tests the performance of GRIM was observed as being slightly better than that of GM. GRIM particularly excelled in the benchmarks involving LANai 4 NI cards. This characteristic can be attributed to the fact that GM is largely targeted for LANai 9 cards and that a number of GM optimizations have to be disabled in order for the LANai 4 cards to function properly. It is important to note that GM is designed to be the most robust and reliable message layer for Myrinet. It is not the intention of this thesis to claim that GRIM is a better message layer than GM. Instead, these measurements are reported for the sake of demonstrating that GRIM provides comparable performance to state-of-the-art message layers such as GM.