An Accelerated Implementation of Portals on the Cray SeaStar

Ron Brightwell Trammell Hudson Kevin Pedretti Keith D. Underwood
Sandia National Laboratories
P.O. Box 5800, MS-1110
Albuquerque, NM 87185-1110

Abstract— **This paper describes an accelerated implementation of the Portals data movement layer on the Cray SeaStar used in the XT3 platform. The current supported implementation of Portals is interrupt-driven and does not take full advantage of the embedded processor on the SeaStar. The accelerated implementation offloads a significant portion of the network protocol stack to the SeaStar, providing significant improvements in network performance. This paper will describe this new implementation and show results for several network micro-benchmarks.**

I. Introduction

The SeaStar high-speed network interface and router in the Cray XT3 contains an embedded PowerPC processor that can be used to offload a significant amount of network protocol processing. However, the current Cray-supported implementation of Portals [1] for the SeaStar does not fully take advantage of this capability. The current implementation is interrupt-driven and requires involvement of the host processor for every message that is received. Sandia has developed a new implementation of Portals that uses the processing capabilities of the SeaStar to its fullest extent, eliminating the interrupt and freeing up the host processor to deliver more compute cycles to applications. This paper describes this new implementation of Portals, referred to by Cray as "accelerated Portals", and provides a performance comparison with the Cray-supported implementation for several micro-benchmarks. We are currently working with Cray to integrate this new implementation into Cray's supported software environment so that it can be available in a future release of Cray software for the XT series of machines.

II. SeaStar Hardware

The embedded processor on the SeaStar is a dual-issue 500 MHz PowerPC 440 processor with independent 32 KB instruction and data caches. The firmware running on the PowerPC is responsible for programming the DMA engines, since programming them via the host processor with accesses over the HyperTransport (HT) is prohibitively slow. On the receive side, the firmware is also responsible for recognizing the start of new messages and processing incoming message headers. The firmware must also recognize and respond to DMA completion events.

To hold the firmware, local state, and handle interactions with the host, the SeaStar has 384 KB of scratch memory. This memory is protected by ECC complete with scrubbing to find and correct errors as they occur. In this context, a certain portion of the network management *must* be done by the firmware running on the network interface. However, the firmware can also be augmented to handle other aspects of the protocol stack. In this paper, we present firmware implementation details for both a host-based mode, with minimal work on the SeaStar, and a NIC-based mode, where most of the Portals processing is offloaded to the PowerPC.

III. SeaStar Firmware

The firmware running on the SeaStar's embedded processor was developed in collaboration with Cray using the C programming language and a standard GNU tool chain for the PowerPC 440. The firmware currently consists of a little more than six thousand lines of C code and approximately two hundred lines of assembly code[1]. When compiled, this results in a 36 KB binary image.

More detailed descriptions of the development tools used and the firmware's internals are given in a previous paper [3]. Here, we provide an overview of the firmware's architecture and then focus on describing our recent work aimed at increasing performance by offloading the network protocol stack to the SeaStar.

A. General Architecture

At the most basic level, the firmware's job is to program the SeaStar's DMA engines to move messages between host memory and the network. The firmware must keep track of, and make progress on, multiple concurrent message transmissions and receptions, and notify the host when each has completed.

On the host, there are some number of client processes that require access to the high speed network. This number varies based on the operating environment. Linux service nodes will typically have many clients (e.g., 10–100 clients), while compute nodes, which run the Catamount lightweight kernel, will have one management client and usually one application process per host CPU core. To keep maintenance

Trammell Hudson is under contract to Sandia via OS Research, Inc.
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

[1]According to Wheeler's SLOCCount tool [2]

tasks manageable, we chose to develop a single firmware image that supports both operating environments.

B. Host-Based Mode

In host-based mode, application-level requests trap into the kernel, which then forwards requests to a single firmware-level mailbox. This approach allows a large number of clients to be supported, since each firmware mailbox and its associated pool of message tracking structures consume a large portion of the SeaStar's 384 KB of memory.

This mode is able to handle transfers that are either physically contiguous or physically discontiguous. For transfers that span physically contiguous regions of memory, the kernel simply passes the starting physical address and length of the message in a command to the firmware. The firmware's messaging machinery then generates the DMA engine commands on-the-fly. In the physically discontiguous case, which is needed to handle requests from Linux processes, the list of DMA engine commands is generated by the kernel and passed to the firmware. When each message transmission or reception is complete, the firmware posts an event to host memory and interrupts the host processor. The OS is then responsible for delivering the event to the appropriate client process.

A more detailed description and performance evaluation of this mode of operation can be found in a previous paper [4].

C. NIC-Based Mode

The goal of this mode is to avoid interactions with the host processor and OS as much as possible, similar to what other high-performance networks have done [5], [6]. The benefits are reduced latency, increased message throughput (messages processed per second), and reduced host overhead related to message processing, making more of the host CPU(s) available for computation. The trade-offs are that the embedded processor is much slower than the host CPU(s) and the amount of SeaStar memory is over two orders of magnitude smaller than the host memory that could be dedicated to message passing. NIC-based mode was developed to evaluate whether these obstacles could be overcome. As with supporting both Linux and Catamount, we chose to implement support for both host- and NIC-based modes in the same firmware image.

In order for a process to use the network, it must call the Portals library initialization routine. Initialization and shutdown requests are forwarded to the firmware via the kernel using the kernel's trusted mailbox. For NIC-based mode, initialization involves allocating SeaStar memory for a new untrusted mailbox and related firmware structures, mapping the mailbox into the process's virtual address space, and providing the firmware with a map of the process's virtual address space (stack, heap, text, read-only data). Once initialization has completed successfully, the process can then initiate further operations by writing directly to its mailbox in SeaStar memory.

Because this user-level mailbox is untrusted, the firmware must carefully inspect all commands that are received on this mailbox. The firmware uses the address map provided by the kernel to insure that data transfers stay within the bounds of the process's address space. This is relatively straightforward for the processes running under the lightweight kernel, since Catamount provides a physically contiguous address space. However, since the SeaStar has no hardware support for virtual-to-physical address mapping, a very limited amount of scratch memory, and a relatively long access time to host memory, supporting operating systems that do not insure a physically contiguous mapping is significantly more complex. Given this extra complexity and the additional performance costs associated with supporting physically discontiguous translations, NIC-based mode only currently works with Catamount.

In addition to protecting the source and destination of data transfers, protection is also needed for a trusted portion of the Portals header that precedes every message. The trusted portion of the header contains information, such as the source node id and process id, that a user-level process should not be able to modify. The steps needed to insure that the trusted header is not modified are complicated by the fact that the SeaStar is only able to send messages from host memory. If this were not the case, the NIC could simply construct the entire header in SeaStar memory and send the header in a single DMA command. Unfortunately, this limitation of the SeaStar means that the trusted portion of the header must be kept in kernel memory and two DMA commands must be used to send the different parts of the message header. However, the added complexity and extra DMA command have been shown to have no significant performance penalty.

Once the firmware recognizes and validates a command written into the mailbox, it performs the requested operation. Some commands, such as memory registration and receive posting, can be handled immediately and the result posted back to the process. Other commands, such as put and get operations, are forwarded to the firmware's messaging machinery. Once the data movement specified by these commands is completed, the firmware generates a completion event and writes it directly into an event queue in the process's memory. The process must eventually poll the event queue to recognize the event. Unlike the host-based implementation, posting an event does not involve raising and processing an interrupt on the host.

The most significant performance benefit of the NIC-based mode comes from the ability to handle message reception autonomously without involving the host processor or OS. For host-based mode, every time a new message arrives, the firmware must raise an interrupt to ask the OS where to deliver it. From a latency perspective, this path is extremely expensive, since the interrupt potentially causes a context switch into the OS and involves several relatively slow round trip accesses across the HT bus. In contrast, the NIC-based implementation has all of the information necessary to determine where in host memory to put incoming messages, effectively bypassing or offloading this responsibility from the OS and application.

When a message arrives, the firmware's messaging machinery passes the header information on to another routine that parses information in the header to determine exactly where in host memory the incoming message should be deposited. Once the ultimate destination of the message is determined, the messaging machinery is given the destination address in host memory and number of bytes to receive and implicitly given the number of trailing bytes to discard, if any. Once message reception is complete, the messaging machinery notifies another routine that is responsible for writing a completion event directly into the host client's event queue in host memory.

D. Flow Control Protocol

As mentioned above, a unique feature of the SeaStar is its hardware support for demultiplexing incoming packets into their appropriate message stream. However, this hardware is limited by its ability to handle only 256 simultaneous message streams from distinct sources. When this capacity is exhausted, long messages from new sources cannot be processed and must be discarded until space is available in the hardware resource. Thus, an end-to-end flow control protocol is needed to recover from lost messages in rare circumstances where the hardware resource is not sufficient to handle the number of concurrent messages being received. We have developed such a flow control protocol for the accelerated implementation of Portals. Called the CAM Overflow Remediation Protocol SystEm (CORPSE), this protocol provides the same reliability and flow-control guarantees as Cray's host-based protocol, called CAM Overflow Protocol (COP).

IV. PORTALS

We have described the Portals implementation for the SeaStar in a previous paper [7]. In this section, we discuss optimizations that were motivated by our initial performance evaluation of the NIC-based mode implementation. We encountered two significant opportunities for optimization – one that involved adding to the Portals specification and one for the way in which our MPI implementation used Portals.

A. MPI Receive Posting

First, performance analysis showed that posting an MPI receive was much slower using NIC-based mode compared to host-based mode. This would often lead to confusing results for our standard suite of micro-benchmarks. Eventually, this issue was traced to the fact that many micro-benchmarks did not insure that MPI receives were pre-posted. NIC-based mode's slower receive posting meant that more messages ended up being unexpected compared to host-based mode, which resulted in degraded performance.

The underlying cause of NIC-based mode's poor receive posting performance was that a round trip to the SeaStar across the HT link is more than an order of magnitude slower than a Catamount system call (approximately 1 μs vs. 65 ns). Posting an MPI receive requires three round trips to the SeaStar. Host-based mode is able to handle this sequence of three calls in the operating system, without involving the firmware, so the overhead of each is roughly equivalent to three system calls. In order to reduce the number of round trips across the HT down to one, we combined these three operations into a single function call, called `PtlPost()`.

When Portals was originally designed, the focus was on identifying a set of general building blocks that could be easily combined to implement a variety of upper level protocols. In this instance, creating a specialized API call that combined several of the simpler building blocks into one aggregate operation led to significant performance advantages for a critical MPI operation. This new call has been added to the official version of the Portals specification.

B. Pre-registration of Memory

The second significant optimization that we made to better support NIC-based mode modified the way MPI used Portals for sending messages. Like the `PtlPost()` optimization, this change was aimed at reducing the number of round trips required to perform a common operation — in this case sending a message. It was also intended to reduce the number of Portals resources that MPI used for sending messages so that more SeaStar memory could be freed up for receiving messages.

As described in [8], the previous implementation of MPI used three different protocols for short, medium, and long messages. In the medium and long message case, MPI would first create a Portals memory descriptor (MD) describing the user's buffer in one operation and then use the new MD to initiate a subsequent put operation. This would result in two HT traversals to initiate each send operation and would also consume an MD for each send, which could potentially consume significant resources for applications that use MPI non-blocking sends.

Catamount allows for an optimization to this approach. Since the regions of a Catamount process are physically contiguous as well as virtually contiguous, the MPI implementation can use only a few MDs to cover the entire address space of a process. During initialization, MPI creates an MD that spans the data region, an MD that spans the stack region, and an MD that spans the heap region. This approach eliminates the need to create an MD for each individual user buffer, since the user buffer is already covered by one of these region MDs. This reduces the number of HT crossings to just the one needed for the put operation.

We expected the impact of this change on the ping-pong bandwidth performance for medium and long messages to be minimal, since an HT crossing is not significant relative to the time needed for the transfer. However, we did expect this optimization to improve message rate and help free up Portals resources allocated from SeaStar memory.

V. TEST ENVIRONMENT

A. Platform

The platform used for our experiments is the 10,368-processor Red Storm machine at Sandia. This machine is a slightly specialized version of the commercial XT3 product. It differs from the XT3 in that the network is not a torus in all three directions. In order to support easily switching portions of the machine between classified and unclassified use, special switching cabinets were created for Red Storm. This capability and the limitation of cable lengths only allow the network to be torus in the z-direction. Each node in Red Storm has a 2.0 GHz Opteron with at least 2 GB of main memory.

B. Benchmarks

We chose several microbenchmarks for our initial evaluation. We began with a simple test of ping-pong latency and bandwidth. For streaming tests, we used a bandwidth benchmark developed by Ohio State University, which posts several messages (64) at the receiver and then sends a stream of messages from the sender. To measure collective performance, we used the Pallas MPI benchmark suite [9] version 2.2.1. Finally, we used a benchmark developed at Sandia to measure the impacts of MPI queue depths on network performance[10].

VI. Results

The results are grouped into three basic categories: traditional point-to-point benchmarks, collective benchmarks, and other data. In each instance, we compare the system with the Portals processing implemented on the host to the system with Portals processing implemented on the NIC. In each case, we also evaluate the impact of adding the CORPSE protocol. For each graph, the left axis graphs performance (either in time or bandwidth), and most graphs include the percentage advantage provided by a NIC-based implementation on the right axis.

A. Basic Point-to-Point Benchmarks

Figure 1 compares the small message latency of the host- and NIC-based Portals implementations. The difference is over 1 μs for extremely small messages and nearly 3 μs for larger messages. This is the advantage of eliminating one interrupt for small messages and two interrupts for larger messages (new message start and message completion). The transition point between 16 bytes and 32 bytes is one of the limitations of the network. The small packet size forces the transition from programmed I/O (PIO) mode transfers to DMA transfers to occur at this point. The addition of the protocol has measurable, but minimal, impact on ping-pong latency as much of the protocol processing can be overlapped with interactions with the host.

The SeaStar has an impressive unidirectional MPI bandwidth (Figure 1) that is attributable to the use of the HT interface to the host. Peak bandwidth is currently over 1.1 GB/s. It is expected that Cray will deliver a second-generation SeaStar in 2006 that will be able to achieve over 2GB/s. While the NIC-based implementation offers very little advantage at extremely large message sizes, at small to moderate message sizes the significant reduction yields major advantages for the NIC-based implementation.

The NIC-based implementation also offers dramatic advantages in streaming bandwidth benchmarks, as shown in Figure VI-A. In the NIC-based implementation, the work is better partitioned between the host and NIC processor. This allows better overall message throughput than the host-based implementation; however, it also begins to introduce limitations as more work is needed on the NIC. The introduction of the protocol, for example, reduces the advantage of the NIC-based implementation because it introduces more work on the NIC. In the host-based implementation, much of this work is overlapped with Portals and MPI processing being done on the host.

Another strong advantage of moving away from bus-based interfaces to the host and toward bi-directional interfaces like HT is the ability to sustain full bi-directional bandwidth. Also shown in Figure VI-A, bi-directional bandwidth on Red Storm achieves twice the uni-directional bandwidth; however, the bandwidth curves suffer somewhat with the introduction of the protocol as there is no longer anywhere to hide the extra processing overhead.

B. Pallas Collective Benchmarks

The Pallas benchmark suite[9] includes benchmarks for numerous collective operations. Rather than include an excessive number of graphs, we have selected four collectives that are relevant to many of the applications at Sandia. The first of these is `MPI_Barrier`. Barrier is a collective that is virtually never *needed* to write a correct MPI program; however, many application developers find it useful for debugging and timing and ultimately leave it in production codes. Since the barrier operation involves a significant number of small messages, it is one of the few benchmarks where the addition of the protocol makes a noticeable difference in performance. Overall, the NIC-based Portals implementation has a significant advantage over the host-based implementation — particularly at larger numbers of nodes.

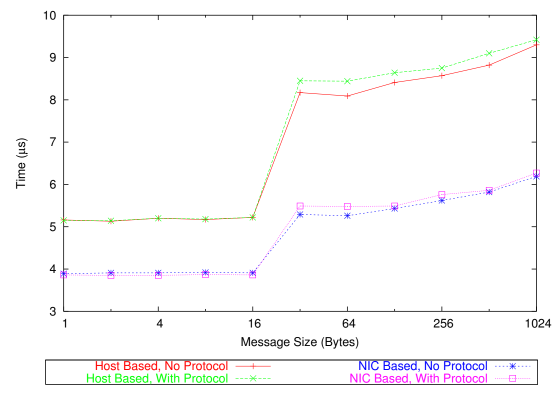
`MPI_Allreduce` and `MPI_Reduce` are also common operations in many of Sandia's codes. Unlike the barrier operation, the reduction operations have data associated with them and require some computation on the host. The results presented in Figure 3 uses 16-byte reductions because that is representative of the data used in Sandia operations (a double-precision number or a double-precision complex number). Moving Portals processing to the NIC has a slightly more significant advantage for `MPI_Allreduce` than for barrier because the `MPI_Allreduce` has work to do on the host that can be overlapped with the protocol processing.

`MPI_Reduce` receives an even greater advantage thanks to its subtle differences from the `MPI_Allreduce` operation. Where `MPI_Allreduce` must reduce a single number and distribute it to all participating processors, a node in `MPI_Reduce` can exit the call as soon as it is done participating in the communication. This means that communications from two consecutive reductions can be overlapped, which leverages the offload provided by the NIC-based implementation. It also means that there is more opportunity to hide the protocol processing, and so `MPI_Reduce` does not suffer a performance penalty from the protocol.

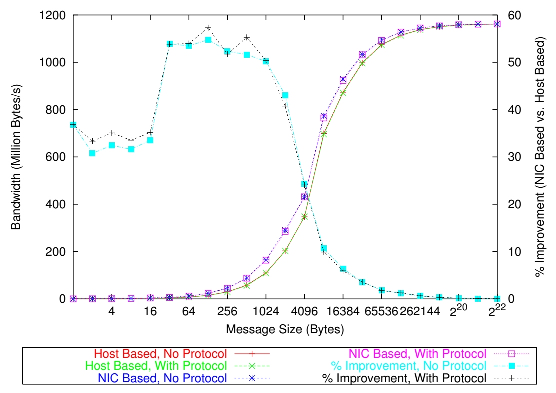
The final collective operation is `MPI_Allgather`. `MPI_Allgather` is an interesting operation in that the number of messages sent *and* the size of the message scale with the number of nodes. Thus, the time grows quadratically with the number of nodes and the time for the host- and NIC-based implementations begin to converge slightly at larger message numbers.

C. Profiling Results

One of the interesting questions to answer is: where does the time go? Figure 4 presents a one-way profile of the ping-pong operations for both the host- and NIC-based Portals

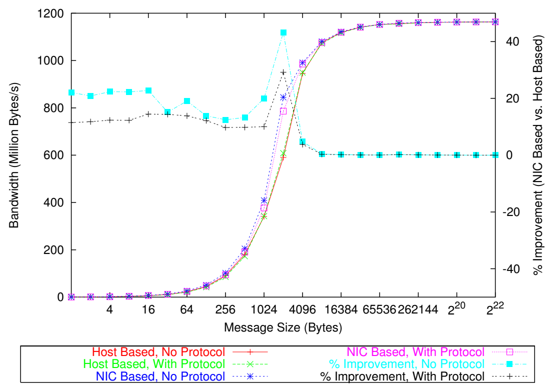


(a)

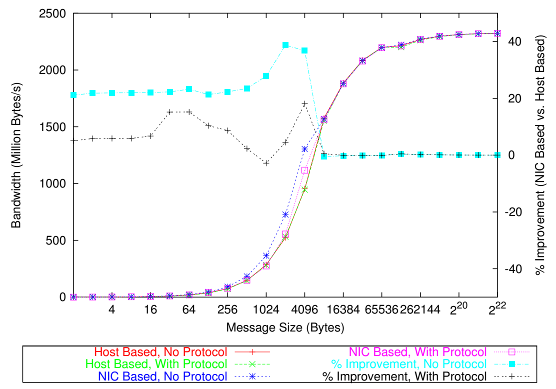


(b)

Fig. 1. MPI latency **(a)** and bandwidth **(b)** performance



(a)



(b)

Fig. 2. MPi Uni-directional **(a)** and bi-directional **(b)** streaming bandwidth performance

implementations. One striking thing to note is that the HT latency is a major contributor to overall latency. While HT is generally much lower latency that other bus interfaces, the architecture of the SeaStar requires that an embedded processor communicate with the host processor through a shared RAM resource. The time for the processors to recognize that a new item has been written is a dominant factor in the HT latency.

The next significant characteristic to note is that the router time and the MPI time are both small slivers of the overall time. The router is highly optimized for HPC (unlike current InfiniBand [11] routers that are an order of magnitude slower, for example) and most of the work for MPI, such as context and tag matching, is included in Portals.

Moving up the bar graphs, the time to notify the host of an Rx event is comparable for both host- and NIC-based implementations, as is the time spent in the "polling loop". The polling loop is the main loop on the NIC that looks for work to be done. It is responsible for polling hardware to check for new events and polling a RAM region used for communicating with the host. The initial firmware implementation used roughly 500 ns per cycle through this polling loop; we have optimized this to less than 130 ns through profiling and analysis of the code paths.

Both transmit and receive operations push more work to the NIC when the NIC-based implementation is used; thus, more time is required to setup transmit and receive operations in the NIC-based implementations. However, the host side Portals work drops dramatically and the interrupt latencies are eliminated when most of the Portals work is moved to the NIC. For the NIC-based implementation, there is still a small amount of time that is not well characterized. It is believed that some of this time is related to the timing granularity on the NIC and some of the overhead added by the protocol.

D. *Protocol Overhead*

The above performance results illustrate the overhead associated with the CORPSE protocol for both the NIC-based and host-based implementations. In order to further characterize the impact of the flow control protocol overhead, we compare the CORPSE protocol to the COP protocol using only the host-based implementation. The results in Figure 6 show a comparison between the host-based implementation of Portals using both CORPSE and COP. There are some minor differences in the code between these two implementations, but these results show that CORPSE has a noticable advantage for small messages for a variety of different communication patterns.

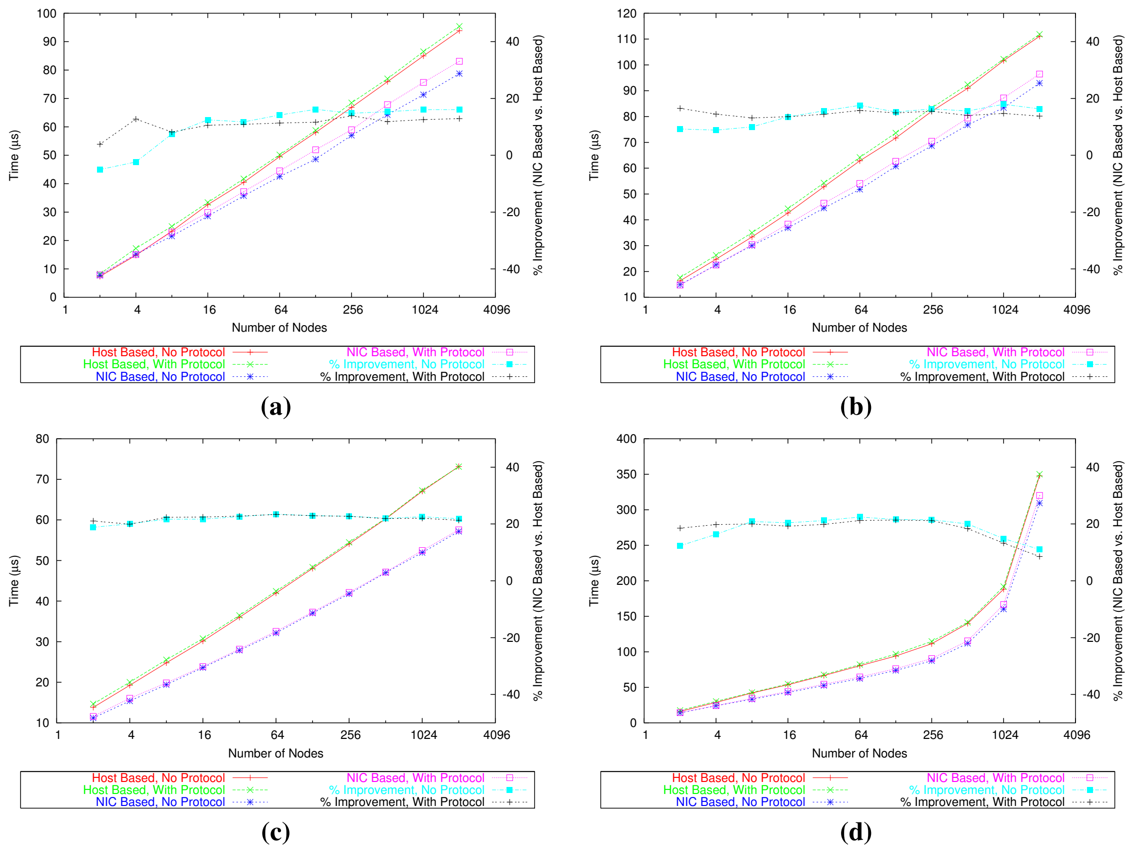


Fig. 3. Pallas MPI benchmark performance for Barrier **(a)**, Allreduce **(b)**, Reduce **(c)**, and Allgather **(d)** (message size = 16 bytes)

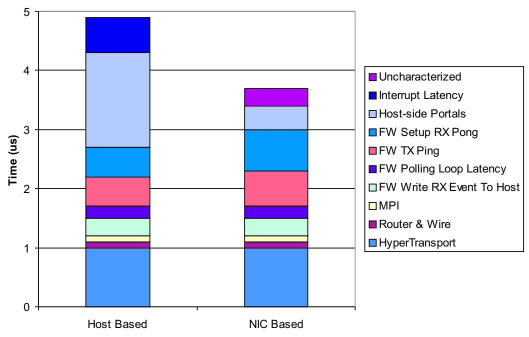
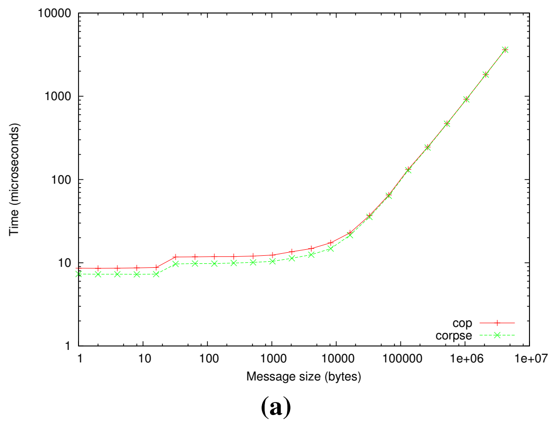


Fig. 4. Profiles of one-way PingPong time

E. Other Issues

Moving processing to the network interface can have significant ramifications for application processing. One of the particular limitations of traditional benchmarks is that they do not consider “real” scenarios. One specific aspect of applications is that they tend to have more than one posted receive. Posted receives are typically kept in a linked list that is walked every time a new message arrives. When that processing is moved from a fast host processor to a slower NIC processor, a long posted receive queue can translate into much high effective network latency. Figure 7 illustrates the difference in latencies as the length of the posted receive queue is increased. The NIC pays a penalty of approximately 30 ns for each item traversed (the fastest NIC in the industry [10]); thus, after traversing a list of 50 items, the NIC-based approach actually loses to the host-based approach.

There are two important caveats attached to this result. Foremost, the nature of the benchmark is inherently more friendly to the more advanced processor in the host. The two major issues are that the host processor has a much larger cache and the host processor has a hardware pre-fetcher that the NIC processor does not. In a benchmark world where the list is strictly ordered, the combination of these two means that host processor is faster than it should be on a per-message basis by at least a factor of five (the host processor should pay a cache miss for every list item). The second caveat is that,



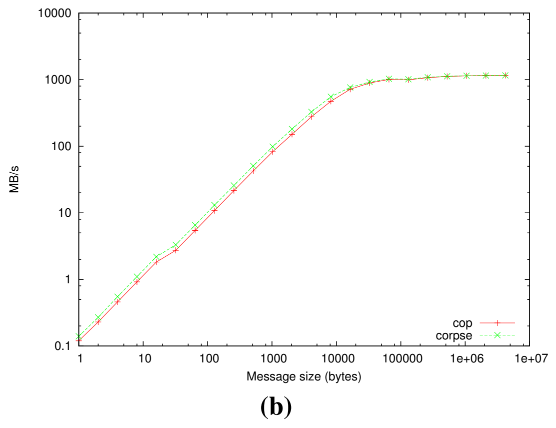


Fig. 5. Pallas MPI PingPong latency **(a)** and bandwidth **(b)** performance

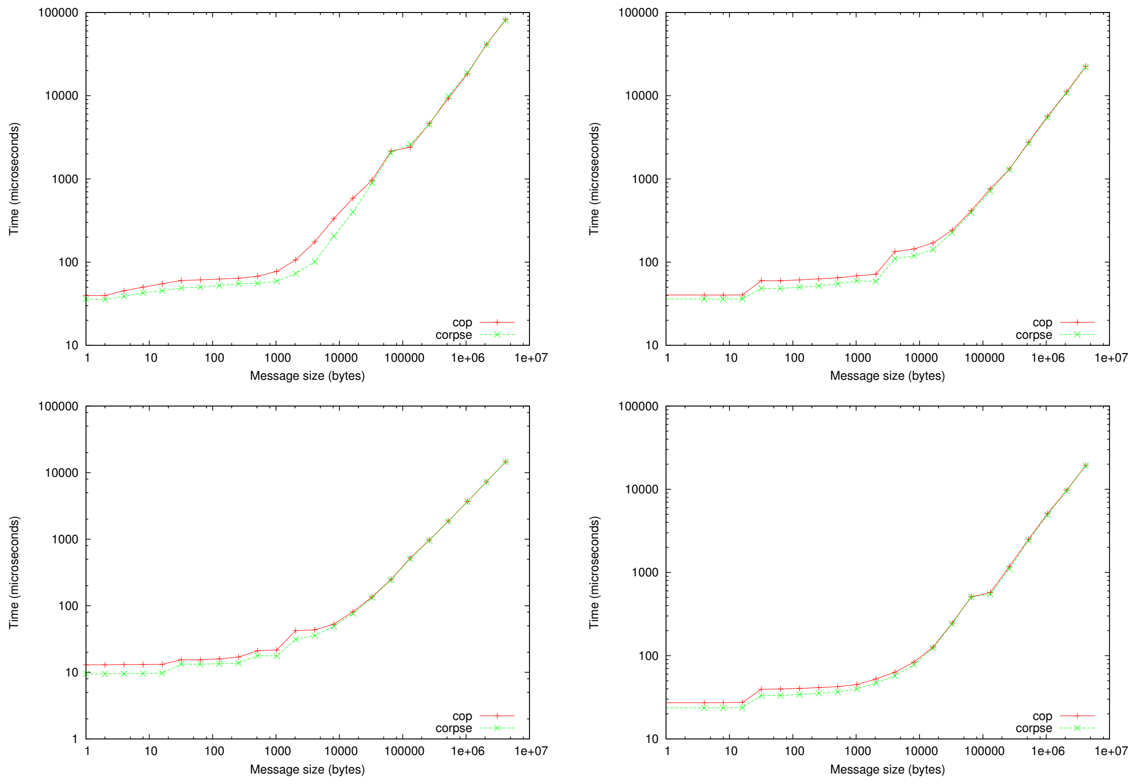


Fig. 6. Pallas MPI benchmark performance for Allgather **(a)**, Allreduce **(b)**, Broadcast **(c)**, and Reduce **(d)** (message size = 16 bytes)

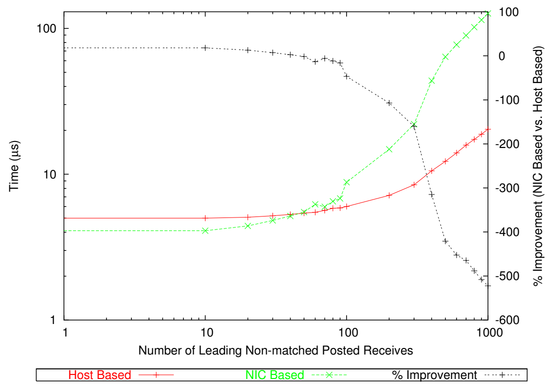


Fig. 7. Effective MPI network latency for various receive queue traversal depths

while some applications use extremely long lists, many have typical list lengths that are thirty items or fewer[12].

VII. Summary

Compared to the initial host-based implementation, the NIC-based implementation of Portals is able to achieve significant performance increases in ping-pong latency and bandwidth as well as streaming bandwidth performance. The ping-pong latency improvement for the NIC-based approach is 26 percent, while ping-pong bandwidth improves between ten and twenty percent for a large range of message sizes. In addition, the NIC-based flow control that has been developed for Portals has a distinguishable, but otherwise minimal, impact on performance.

We have also described optimizations to upper-level protocols that were driven by initial performance results from NIC-based mode. In particular, the relatively slow accesses across the HT bus to SeaStar memory motivated an additional Portals API call and a change to the MPI implementation to reduce the number of HT crossings.

VIII. Future Work

The Cray XT3 is a system that is continuously improving. The protocol has opportunities for improvement that we expect to implement in the near future. For example, a backoff scheme to better manage a flood of traffic to a single node is currently being implemented. In addition, performance optimizations that leverage the 500 MHz embedded PowerPC, such as offloaded collective operations, are being considered.

We are also considering a rendezvous protocol on the PowerPC for long messages. The current MPI implementation uses eager sends for all messages and is occasionally forced to drop long messages at the receiver if a matching posted receive is not found. Implementing rendezvous protocol in the NIC will allow support for true independent progress in MPI while eliminating the risk of potentially retransmitting large messages.

We are also planning an in-depth analysis of the impact of these additions and changes on real applications at scale. Several of these enhancements have the potential for significant increases in application performance and scalability.

And, we are currently working with Cray to integrate this new implementation of Portals into their code base so that it may be available in a future release.

IX. Acknowledgments

The authors gratefully acknowledge the work of the members of the Scalable Computing Systems and Scalable Systems Integration departments at Sandia, especially Jim Laros and Sue Kelly.

References

[1] R. Brightwell, W. Lawry, A. B. Maccabe, and R. Riesen, "Portals 3.0: Protocol building blocks for low overhead communication," in *Proceedings of the 2002 Workshop on Communication Architecture for Clusters*, April 2002.

[2] *SLOCCount*, David A. Wheeler, available from http://www.dwheeler.com/sloccount.

[3] K. T. Pedretti and T. Hudson, "Developing custom firmware for the Red Storm SeaStar network interface," in *Cray User Group Annual Technical Conference*, May 2005.

[4] R. Brightwell, K. Pedretti, and K. Underwood, "Initial performance evaluation of the Cray SeaStar interconnect," in *Proceedings of the 13th IEEE Symposium on High-Performance Interconnects*, August 2005.

[5] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics network: High-performance clustering technology," *IEEE Micro*, vol. 22, no. 1, pp. 46–57, January/February 2002.

[6] Myricom, Inc., "Myrinet Express (MX): A high performance, low-level, message-passing interface for Myrinet," July 2003. [Online]. Available: http://www.myri.com/scs/MX/doc/mx.pdf

[7] R. Brightwell, T. Hudson, K. Pedretti, R. Riesen, and K. Underwood, "Implementation and performance of Portals 3.3 on the Cray XT3," in *Proceedings of the 2005 IEEE International Conference on Cluster Computing*, September 2005.

[8] R. Brightwell, "A comparison of three MPI implementations for Red Storm," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 12th European PVM/MPI Users' Group Meeting, Sorrento, Italy, September 2005 Proceedings*, ser. Lecture Notes in Computer Science, B. D. Martino, D. Kranzlmuller, and J. Dongarra, Eds., vol. 3666. Springer-Verlag, 2005, pp. 425–432.

[9] *Pallas MPI Benchmarks*, http://http://www.pallas.com/e/products/pmb/index.htm.

[10] K. D. Underwood and R. Brightwell, "The impact of MPI queue usage on message latency," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, Montreal, Canada, August 2004.

[11] *http://www.infinibandta.org*, Infiniband Trade Association, 1999.

[12] R. Brightwell and K. D. Underwood, "An analysis of NIC resource usage for offloading MPI," in *Proceedings of the 2004 Workshop on Communication Architecture for Clusters*, Santa Fe, NM, April 2004.