Low-Latency Message Passing on Workstation Clusters using SCRAMNet[1] [2]

Vijay Moorthy, Matthew G. Jacunski, Manoj Pillai,Peter, P. Ware, Dhabaleswar K. Panda,
Thomas W. Page Jr., P. Sadayappan, V. Nagarajan† and Johns Daniel†

Dept. of Computer and Information Science,
The Ohio State University
Columbus, OH 43210
moorthy|jacunski|pillai|ware|panda|page|saday@cis.ohio-state.edu

† SYSTRAN Corporation
4126 Linden Avenue
Dayton, OH 45432
nagu|jdaniel@systran.com

Abstract

Clusters of workstations have emerged as a popular platform for parallel and distributed computing. Commodity high speed networks which are used to connect workstation clusters provide high bandwidth, but also have high latency. SCRAMNet is an extremely low latency replicated non-coherent shared memory network, so far used only for real-time applications. This paper reports our early experiences with using SCRAMNet for cluster computing. We have implemented a user-level zero-copy message passing protocol for SCRAMNet called the BillBoard Protocol (BBP). The one way latency for sending a 4-byte message between two nodes using the BBP is measured to be as low as 7.8 μs. Since SCRAMNet supports hardware level replication of messages, it is possible to implement multicast with almost the same latency as point-to-point communication. Using the BBP, the latency for broadcasting short messages to 4 nodes is measured to be 10.1 μs and the latency for a 4-node barrier is measured to be 37 μs. We have also built an MPI library on top of the BBP which makes use of multicast support from the BBP. Our results demonstrate the potential of SCRAMNet as a high performance interconnect for building scalable workstation clusters supporting message passing.

1. Introduction

Workstation clusters have emerged as a popular, cost-effective platform for parallel and distributed computing[1, 2]. Unlike conventional parallel computers, workstation clusters use commodity interconnection technologies such as ATM, Fast Ethernet and Fiberchannel which provide high bandwidth, but also have high latency due to high software overhead. There is considerable ongoing research for developing low latency networks such as Myrinet [4] and messaging libraries such as FM [12] and U-Net [14].

SCRAMNet (Shared Common RAM Network) [5, 13] is a replicated non-coherent shared memory network. It was initially developed for systems to execute real-time applications that require low latency communication, such as aircraft simulators, industrial process control, virtual reality, telemetry and robotics. For these applications, SCRAMNet has been almost exclusively used for shared memory programming. Recently, synchronization mechanisms have been developed for SCRAMNet [10], but little work has been done for providing message passing support on SCRAMNet.

In this paper, we present our early experiences with building a low latency message passing system for workstation clusters using SCRAMNet. We have developed a message passing protocol called the BillBoard Protocol (BBP). It is a zero copy, lock-free protocol. The BBP Application Programming Interface (API) provides point-to-point as well as collective communication primitives. We have implemented an MPI library on top of the BBP API. In most MPI implementations collective operations are built on top of point-to-point primitives. In our implementation MPI collective operations (broadcast and barrier) are built on top of the BBP API collective primitives.

This paper is organized as follows. Section 2 provides a brief overview of SCRAMNet technology. Section 3 describes the implementation of the BBP. Section 4 describes the implementation of th MPI implementation on top of the BBP. Section 5 desribes performance evaluation results at the BBP and MPI layers. Section 6 describes related work done elsewhere. Section 7 presents our conclusions.

2. SCRAMNet Networking

This section briefly describes SCRAMNet networking. It is described in detail in [5, 13].

SCRAMNet consists of network interface cards (NICs) with memory banks connected via a dual ring. The NICs can be plugged into standard I/O buses such as PCI and S-Bus. When a workstation writes to a memory location on its NIC, the write is reflected at the same location on the memory banks on the other NICs connected to the network. The memory is shared, as a write by one node on the ring can be read by all others. However, the memory is not coherent, that is, if two nodes concurrently write to the same location, all the nodes may not see the writes in the same or-

[1]This research is supported by an NSF-SBIR Grant DMI-9761318.
[2]SCRAMNet is a registered trademark of SYSTRAN Corporation.

der. Since SCRAMNet uses a ring topology, the time taken for a write to propagate throughout the ring is bounded and predictable.

Physically, SCRAMNet is arranged as a ring of up to 256 nodes. The ring is normally made of fiber-optic cable. For shorter distances, coaxial cabling can also be used. For systems larger than 256 nodes, a hierarchy of rings can be used. The data transfer latency between two neighboring nodes is 250-800 nanoseconds, depending on the transmission mode. SCRAMNet normally uses fixed 4-byte packets for transmission. This mode provides a maximum throughput of 6.5 MBytes/sec. Variable length packets with lengths varying from 4 bytes to 1 KByte can also be used. In this mode, the maximum throughput is 16.7 MBytes/sec but latency is also higher.

No driver software is required to use these NICs as the data transfer is handled at the hardware level. Most OS's allow the shared memory on the NIC to be mapped to the virtual address space of the workstation. Once this mapping is done, any write to the memory on the NIC is automatically and immediately propagated to the other NICs on the ring network. In other words, data can be transferred from one node to another using a single store instruction, or an assignment statement in a high level language. For larger data transfers, programmed I/O or DMA can be used.

Shared memory and message passing are the two programming models available for parallel programming. Applications which require shared memory tend to use physically shared memory for communication. For large systems physically shared memory is very difficult to build. Larger systems, therefore use message passing. However, message passing using conventional LAN networking technology has very high communication overhead. Distributed Shared Memory (DSM) systems try to provide an abstraction of shared memory on top of message passing systems. For a DSM system to work, it has to provide cache coherence. Scaling DSM systems while maintaining cache coherence, however, is a difficult problem. SCRAMNet offers a middle path. It has a limited amount of non-coherent shared memory, but it is scalable. SCRAMNet does not have very high bandwidth but it has extremely low latency. In addition, message latencies are bounded and there is no overhead of protocol information to be added on messages.

3. The BillBoard Protocol and the API

This section briefly describes the message passing protocol that we developed for SCRAMNet called the BillBoard Protocol. The protocol and the associated API are described in detail in [8]. The BillBoard Protocol derives its name from the manner in which communication is done using it. When a process has to send a message, it posts the message at one place, where it can be read by one or more receivers. This method of transmission is akin to advertising on a billboard, hence the name. This protocol has very low latency because no process has to wait for a lock as any location is written by exactly one process. From the processor point of view, it is a zero copy protocol as messages are moved directly from user-space buffers to SCRAMNet (NIC) memory.

The SCRAMNet memory is divided equally among the participating processes. Each process' memory is divided into a data partition and a control partition. Messages are written to buffers allocated in the data partition. The control partition contains MESSAGE flags, which indicate waiting messages from other processes; ACK flags, which indicate waiting acknowledgements; and buffer descriptors, which contain offset and length of messages in the data partition.

Let us see how communication occurs using the BBP. Suppose process P_1 wants to send a message to process P_2. First P_1 tries to allocate a buffer in its data partition. If there is sufficient space, a message buffer is allocated for it [3]. P_1 then writes a message buffer descriptor in its control partition and writes the message in the allocated buffer. P_1 also sets a MESSAGE flag in P_2's control partition corresponding to P_1 and the allocated message buffer. During the receive operation, P_2 reads the MESSAGE flags from its control partition in the shared memory and compares it with previously stored values. Any changes indicate newly arrived messages, and their bit positions indicate the message buffer in the senders' data partition. To receive the message, P_2 reads the offset and length of the message from P_1's control partition. Then it reads the appropriate buffer from P_1's data partition and toggles P_2's acknowledgement flag for that buffer in P_1's control partition. The message transfer is then complete.

Since the data partition of every process is visible to all others, this protocol can be very easily extended for collective operations. To send a multicast or a broadcast message, a process allocates a buffer in its data partition, writes the message to the allocated buffer, and then sets MESSAGE flags for more than one receiver (all the other processes in case of broadcast). Each extra receiver, adds only the cost of writing one more word to SCRAMNet memory at the sender. Unlike point-to-point based algorithms for multicast such as the binomial tree algorithm, this type of multicast is a single step algorithm, which means that potentially, all the receivers could receive the multicast message simultaneously.

The BBP API is quite simple. It provides 5 functions for initialization, (`bbp_init`), sending (`bbp_Send`), receiving (`bbp_Recv`) and multicasting messages (`bbp_Mcast`) and checking for newly arrived messages (`bbp_MsgAvail`). The complete specification of the API can be found in [8].

[3]If a buffer cannot be allocated garbage collection is first done to free up space in the data partition and then a buffer is allocated.

4. Implementing MPI over SCRAMNet

MPI is a popular standard for writing portable parallel programs. This section describes our implementation of MPI, which is built on top of the BillBoard API. Our implementation of MPI is a derivative of MPICH [6], a publicly available implementation of MPI from Argonne National Labs. Most research implementations of MPI as well as many commercial MPI implementations are derivatives of MPICH. The architecture of MPICH is briefly described below. For details, readers are referred to [6, 11].

MPICH has a 4-layered architecture. The top two layers contain the bindings for the MPI functions. The lower of these two layers, the point-to-point binding layer sits on top of the Abstract Device Interface (ADI) layer. Though complex functions such as collective communication operations and topology mapping are built on top of the basic point-to-point layer, hooks are provided in MPICH to extend it so that these more complex operations can directly use any additional functionality provided by the underlying device.

The ADI is the key to the portability of MPICH. Some devices may provide limited functionality while others may provide more complex functionality. The ADI can be implemented on top of any device and provides a rich, uniform interface. One implementation of the ADI is in terms of a lower level interface called the channel interface. Because the channel interface is a much more rudimentary interface, it provides the fastest route to porting MPICH to a new system. We have used this approach in our implementation and developed a SCRAMNet Channel layer device which is a minimal implementation of the Channel Interface.

In MPICH, all collective operations are built on top of point-to-point functions. We have modified it to use BBP multicast (`bbp_Mcast`) to implement the MPI_Bcast and MPI_Barrier.

In our implementation of the MPI_Bcast, the process that is the root (or sender) of the MPI_Bcast determines the processes in the group. It then uses the multicast operation in the BBP API to broadcast the data to each process in the group. All receiving processes wait for a message from the root process. Our implementation of broadcast is not synchronizing, that is, the root of the broadcast does not wait for other processes to arrive at the MPI_Bcast call. Since the BBP API provides in-order delivery, multiple MPI_Bcast operations are matched in order.

In our implementation of the MPI_Barrier, the process with rank 0 acts as the co-ordinator. It waits for a null message from every other process in the group associated with the communicator. After it has received messages from all processes in the group, it uses the multicast operation in the BBP API to send a null message to all other processes in the group.

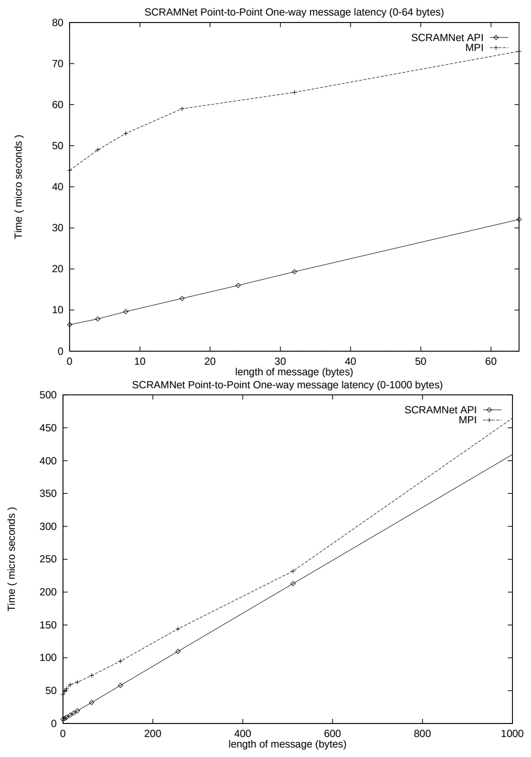

Figure 1. One-way Message latency for SCRAMNet Using BillBoard API and MPI

5. Performance evaluation

This section describes the performance of our message passing system at both API and MPI levels. The measurements were taken on a network of 4 Dual Pentium II 300 MHz SMP boxes connected by SCRAMNet and running LINUX version 2.0.30. The results are also compared with comparable implementations on other contemporary technologies namely, Fast Ethernet, ATM and Myrinet, on the same testbed.

Figure 1 shows the one-way latencies for sending messages on SCRAMNet at the API level and MPI level. For small messages, the latency is very small. At the API layer, a 4-byte message can be sent in 7.8 μs. At the MPI layer a 4-byte message can be sent in 49 μs. A zero byte message can be sent at the API and MPI layers in 6.5 μs and 44 μs respectively. It can be observed that the MPI layer only adds a constant overhead to the API layer latency.

Figure 2 compares the performance of SCRAMNet with Fast Ethernet, ATM and Myrinet at the API layer. The measurements for Fast Ethernet and ATM were taken with TCP/IP. The measurements for myrinet were taken with the native myrinet API as well as TCP/IP. SCRAMNet does better than Fast Ethernet for message lengths up to several thousand bytes. It is better than ATM for message lengths less than 1000 bytes. It has lower latency than Myrinet using myrinet API for messages shorter than approximately 500 bytes.

Figure 3 compares the performance of SCRAMNet with

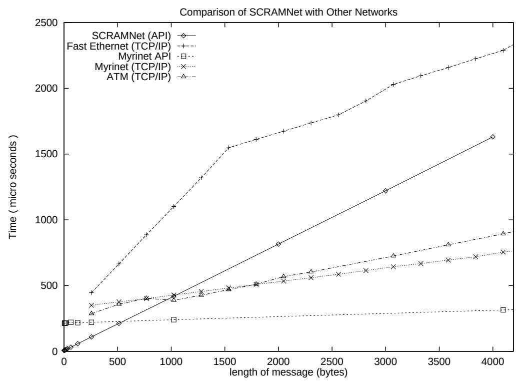

Figure 2. Comparison of SCRAMNET with other networks at the API layer

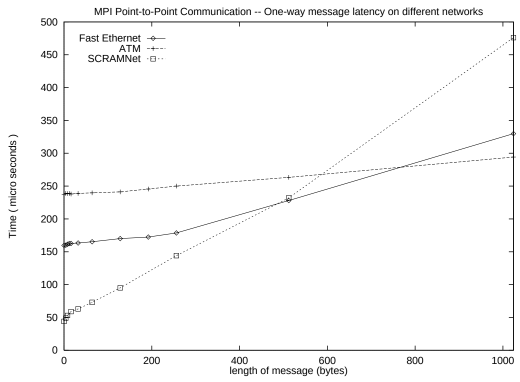

Figure 3. Comparison of SCRAMNET with other networks at the MPI layer

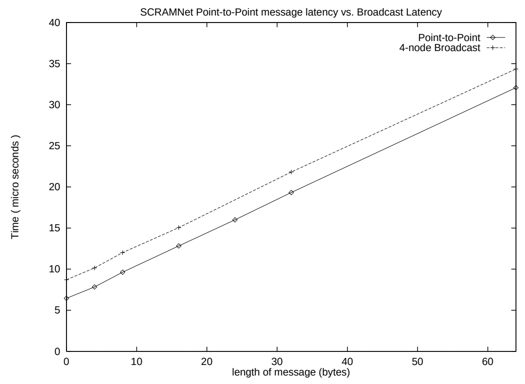

Figure 4. Comparison of SCRAMNET point-to-point and broadcast communication

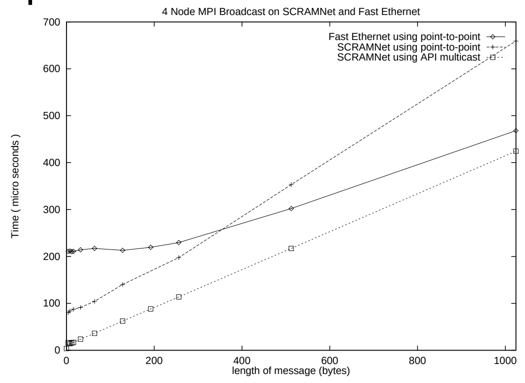

Figure 5. Comparison of MPI broadcast on SCRAMNet and Fast Ethernet

Fast Ethernet and ATM at the MPI level. The measurements shown for Fast Ethernet and ATM were taken using MPICH with TCP/IP. As in the API layer, we see that SCRAMNet does well for small messages and with each of the other two networks, there is a threshold beyond which the other network does better. SCRAMNet is faster than Fast Ethernet for messages shorter than approximately 512 bytes and faster than ATM for messages shorter than approximately 580 bytes.

Figure 4 shows the time taken for a point-to-point message and for a 4-node broadcast. It can be observed that a 4-node broadcast adds very little overhead to a unicast message.

Figure 5 compares the performance of MPI broadcast on Fast Ethernet and SCRAMNet. Two implementations of MPI_Bcast for SCRAMNet are shown - one using the standard point-to-point based algorithm in MPICH, and one using the API level multicast. Comparing the point-to-point based broadcast, we see a trend similar to that for point-to-point communication. For short message lengths, SCRAMNet does better than Fast Ethernet, but for messages longer than approximately 450 bytes, Fast Ethernet does better. Our implementation of MPI_Bcast using the API level multicast is much faster. In this experiment, up to message lengths of 1 Kbyte, it is faster than Fast Ethernet.

Figure 6 shows the performance of Barrier on SCRAMNet. As in the case of MPI_Bcast two implementations of MPI_Barrier are shown, one using the standard MPICH point-to-point based algorithm, and one using the API level multicast. MPI_Barrier is more expensive on Fast Ethernet and ATM than SCRAMNet. For instance, a 3-node barrier takes 554 μs on Fast Ethernet and 660 μs on ATM, while it takes only 179 μs on SCRAMNet using the point-to-point algorithm. The implementation using the API level multicast takes only 37 μs.

6. Related work

There has been recent research on low latency communication using using shared memory interconnects, mainly DEC Memory Channel [9] and Princeton SHRIMP [3].

In Memory Channel, the virtual address space of a process is mapped to the virtual address space of another process so that a store instruction in the physical memory of one system is reflected in the physical memory of another. Sharing of memory in Memory Channel is one-to-one. The Princeton SHRIMP multicomputer uses the routing network of Intel's Paragon systems and recently, Myrinet. It is not general purpose, as it requires the cache to snoop on DMA writes as well.

There is also work being done to provide fast collec-

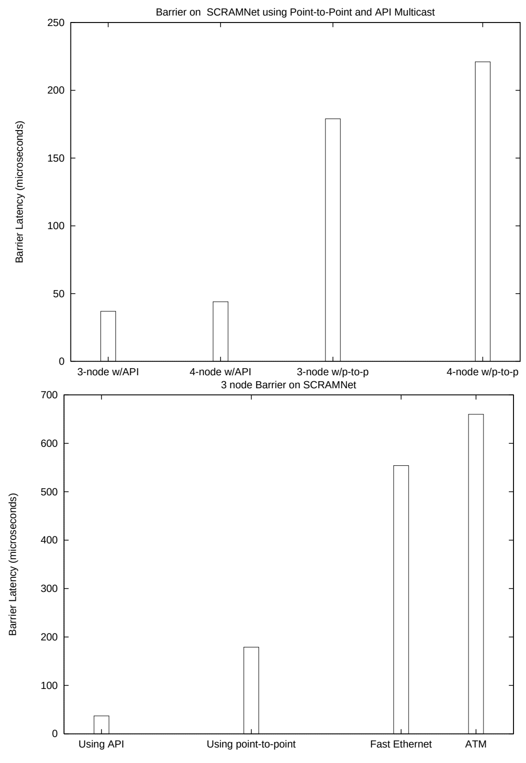

Figure 6. Performance of MPI Barrier on SCRAMNet

tive communication on workstation clusters using dedicated hardware support. PAPERS (Purdue's Adapter for Parallel Execution and Rapid Synchronization) [7] use specialized networks, called Bitwise Aggregate Networks, for fast collective communication. In our research, we have tried to use a generic approach to support both point-to-point and collective communication efficiently on workstation clusters using SCRAMNet.

7. Conclusions

We have presented an implementation of a low latency message passing layer for workstation clusters using SCRAMNet. This implementation provides a basic message passing API on SCRAMNet. We have also implemented MPI on top of the API. For short messages, SCRAMNet has been shown to perform better than other networks (Fast Ethernet, Myrinet and ATM) for point-to-point communication as well as broadcast and barrier.

SCRAMNet has low latency, but it does not have high bandwidth. It is possible to implement collective communication operations with little overhead over point-to-point operations. We conclude that SCRAMNet has characteristics complementary to those of networks usually used in clusters. This makes SCRAMNet a good candidate for use with a high bandwidth network within the same cluster. We are working on using SCRAMNet together with other networks such as Myrinet and ATM to design efficient communication subsystems for workstation clusters which have low latency as well as high bandwidth.

We are currently working along two directions to reduce MPI point-to-point latency even further. The first direction is to remove the Channel Interface layer by creating an Abstract Device Interface layer directly on top of the BillBoard API. The second direction is to incorporate an interrupt mechanism to enhance the communication performance for long messages. Currently, our MPI implementation uses polling to check for incoming messages. Polling requires memory access across the I/O bus which increases the receive overhead. Also, currently each layer has to manage received message queues due to the absence of an interrupt mechanism in the BBP layer. With these directions, we plan to develop a faster framework for message passing on SCRAMNet.

A more detailed version of this paper can be found in [11]. Additional information is available at *http://nowlab.cis.ohio-state.edu/* and *http://www.cis.ohio-state.edu/˜panda/pac.html*.

References

[1] T. Anderson, D. Culler, and D. Patterson. A Case for Networks of Workstations (NOW). *IEEE Micro*, pages 54–64, Feb 1995.

[2] D. J. Becker et al. Beowulf: A Parallel Workstation for Scientific Computation. In *International Conference on Parallel Processing*, 1995.

[3] M. A. Blumrich, C. Dubnicki, E. W. Felten, and K. Li. Protected, User-level DMA for the SHRIMP Network Interface. In *IEEE Int'l Symposium on High Performance Computer Architecture (HPCA-2)*, pages 154–165, 1996.

[4] N. J. Boden, D. Cohen, et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, pages 29–35, Feb 1995.

[5] T. Bohman. Shared-Memory Computing Architectures for Real-Time Simulation-Simplicity and Elegance. Technical report, Systran Corporation, 1994.

[6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Technical report, Argonne National Laboratory and Mississippi State University.

[7] R. Hoare, H. Dietz, T. Mattox, and S. Kim. Bitwise Aggregate Networks. In *Symposium on PODC*, 1996.

[8] M. G. Jacunski et al. Low-Latency Message Passing for Reflective Memory Networks. In *Workshop on Communication, Architecture and Applications for Network-Based Parallel Computing (CANPC)*, 1999.

[9] J. V. Lawton, J. J. Brosnan, et al. Building a High performance Message-passing system for MEMORY CHANNEL Clusters. *Digital Technical Journal*, 8(2):96–116, 1996.

[10] S. Menke, M. Moir, and S. Ramamurthy. Synchronization Mechanisms for SCRAMNet+ Systems. In *Proceedings of the 16th Annual Symposium on the Principles of Distributed Computing*, July 1998.

[11] V. Moorthy et al. Low Latency Message Passing on Workstation Clusters using SCRAMNet. Technical Report OSU-CISRC-10/98-TR42, October 1998.

[12] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[13] Systran Corporation. *SCRAMNET+ Network*, 1994.

[14] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating Systems Principles*, 1995.