A Stream Protocol Implementation for an SCI-based Cluster of Workstations

Jørgen S. Hansen Povl T. Koch Eric Jul

Department of Computer Science, University of Copenhagen (DIKU)

Universitetsparken 1, 2100 Copenhagen, Denmark

`cyller,koch,eric@diku.dk`

Abstract

New high performance memory mapped networks such as SCI provide both low latency and high bandwidth. Many network servers can benefit from this performance, but use standard protocols such as TCP/IP which usually carry a high software overhead. We have implemented a prototype called SciStream, which exploits the low level performance of SCI cluster hardware, but retains compatibility with legacy protocols. The SciStream protocol is implemented at kernel level and is integrated with the TCP/IP subsystem of a general operating system. The implementation combines a number of optimizations and provides traditional protection and the ability to handle many concurrent processes. We present early performance measurements that show end-to-end latencies around 8.4 microseconds and bandwidths up to 72 Mbytes/s.

1 Introduction

High performance network servers can no longer be implemented using a single computer—their service requirements are substantial—and growing. We base high performance network servers on clusters of multiprocessor workstations connected by a high performance network, namely a Scalable Coherent Interface (SCI) network. Because network server applications typically base network communication on TCP/IP, the high bandwidth and low latency of SCI networks are not readily available to such applications. Furthermore, because a cluster-based network server often uses a mixture of networking technologies, providing uniform access to all these technologies greatly simplifies both application development and server maintenance. This paper concerns the development of such uniform access to SCI networks and regular IP based networks. This is done providing a stream protocol compatible with TCP for SCI networks, and integrating it with the TCP/IP subsystem of a general operating system.

The rest of this paper is organized as follows. Section 2 provides a short introduction to SCI and Dolphin's PCI-SCI adapter. Section 3 describes how access to the SCI network is integrated with the operating system and the applications. Section 4 gives a description of our SCI based stream protocol. Finally, Section 5 evaluates the performance of the stream protocol on our testbed consisting of regular workstations, the Linux operating system, and Dolphin's PCI-SCI adapters.

2 Dolphin's PCI-SCI Adapter

Dolphin's PCI-SCI adapter [2] is based on the IEEE SCI standard [4] which specifies the implementation of a coherent shared memory abstraction and a low-level message passing interface. Cache coherence is optional and typically it is not implemented by I/O-based SCI implementations. The PCI-SCI adapter provides mechanisms to map remote memory, to send and receive low-level messages, to generate remote interrupts, and to provide atomic operations on shared memory. SCI is based on a ring topology and Dolphin's implementation is based on 400 Mbytes/s bi-directional point-to-point links and SCI switches which allows the interconnection of rings.

Nodes share physical memory through remote memory mappings. A node accesses physical memory on a remote node by mapping it into the address space of its local PCI bus. Memory accesses to that memory range are then converted into accesses to the remote memory. User level processes can access remote memory by mapping the specific address range of the PCI bus into its virtual address space. The remote writes of the PCI-SCI adapter are not guaranteed to be FIFO—writes across a memory range are split into a number of FIFO streams. Thus, writes to different streams may appear in remote memory out of order. To compensate for this problem, facilities are provided to wait for all writes to be performed.

3 Integration of Specialized Network Interfaces

We implement a stream protocol (SciStream) compatible with TCP on top of our SCI network, as to give the applications of a cluster-based network server access to the network while retaining the most common user interface for such applications. In this section, we deal with the issues of integrating this specialized stream protocol with the operating system and the applications. The main issue is whether the protocol should be implemented at user or at kernel level. Furthermore, we deal with how to add the implementation to the operating system, and how the implementation is made available to the application programmers.

Our protocol has been implemented at the kernel level because doing so provides protection from malfunctioning or malicious user level processes, and allows easy coordination of multiple processes. Furthermore, full access to the virtual memory and networking subsystems of the kernel is possible. The cost of implementing at the kernel level is an increase in latency due to system calls.

We have added SciStream to the Linux kernel by modifying the networking subsystem to allow dynamic loading and removal of protocol implementations at runtime. This is done by using the loadable kernel modules in Linux and by making a minor extension to the standard interface for the INET protocol family that the networking subsystem already implements.

The protocol implementation is accessible to the application programmer through the address space of the network. In the TCP/IP subsystem, the special purpose implementation takes over part of the IP address space. All communication to these IP addresses uses the special purpose implementation as long as the kernel module is loaded. When the kernel module is unloaded, the regular TCP/IP protocol is used. Active connections will not survive this reconfiguration, but new connections will work. This improves fault tolerance with respect to errors in the special purpose network—in case of errors it is simple to revert to the regular protocol.

4 An SCI-based Stream Protocol

Our SciStream implementation uses a circular buffer of pages for communication, and employs several techniques to reduce latency and data copying. In the following , we describe both the basic operation of the stream protocol and several optimizations. Currently SciStream uses only programmed I/O but DMA support is now available in the Linux SCI driver and we will explore this possibility later.

4.1 Basic Operation

The receiver allocates a series of pages in shared memory to act as receive buffers. Furthermore, both sender and receiver allocates control data structures in shared memory to direct the data transmission. For each page these data structures include both a head pointer, a tail pointer, and a status flag indicating whether or not the page may be used by the sender. This approach allows the receiver to perform remapping of pages by looking at the flag, and it allows the sender to skip part of a page, if this improves the alignment of the page with regards to page remapping. To be able to dynamically change the size of the circular buffer, each page has an index. This index is also used by the receive posting and small packet optimizations, as they require that memory not part of the circular buffer can be inserted into the data stream.

To avoid unnecessary interrupts, we use the lock mapping facility of the PCI-SCI adapter. Using this mapping reads from remote memory are converted to an atomic fetch and increment on the remote memory, and writes to the remote memory are converted to atomic increments. Furthermore remote writes have the property that if the remote memory location written to was zero before the increment, an interrupt is generated on the remote node.

4.2 Receive Posting Optimization

Receive posting [9] is used to improve the performance of request/reply communication patterns. When a process transmits a request, e.g., to a server, the process will typically block while waiting for the reply. When using receive posting the server will also be told about the buffer pages allocated to the reply thus allowing the server to write the reply directly to the final memory destination. This reduces the latency of request/reply communication but works only when the receiver of the reply is blocked.

In SciStream, when a receiver blocks waiting for a reply, it posts the receive buffer by pinning the buffer pages in memory and by writing their addresses to a special area in the senders control block together with the starting offset within the pages and the size the buffer. When the sender transmits the reply, it examines the control block and if a valid buffer is present, this buffer is used for the transmission. As there is a delay from a request is sent until the receiver blocks waiting for the reply, it is possible for the sender of the reply to start transmitting while the receive posting is being performed, and the receive posted buffer must be invalidated. The validation is done by attaching information about the last write seen by the receiver to each receive posting. This allows the sender to immediately validate a receive posting. Whether the posted buffer was used or not, the receiver must still unpin the pages after its next receive.

4.3 Page Remapping Optimization

When the receiver does not block waiting for data, e.g., when using asynchronous I/O, receive posting cannot be used to avoid data copying. If the received data is large enough, it can be worthwhile to remap the pages containing the data instead of copying the data. Such copying requires that the receivers buffer and the received data have the same alignment. To make this likely, we try to coordinate how large packets are aligned by the sender and the receiver either by having the receive buffers be page aligned or by trying to determine a common offset within a page for the buffers used by the receiver. This offset can stem from either the allocation strategy of the memory allocator or because the receiving process uses the same buffer for all receives.

When a sender transmits a large packet (specified by a remapping threshold in the implementation), it will start on the predetermined alignment, skipping any partially full pages, if necessary. This may reduce the utilization of the receive buffer by creating internal fragmentation. In case a receiver determines that a sender's attempt to align the data transmission is unsuccessful, the optimization can be disabled by the receiver on a per connection basis to avoid such fragmentation.

4.4 Small Packets

The two previous optimizations dealt with avoiding data copying when sending large packets. When sending small packets, the data copying overhead is small and other factors dominate latency. As described in Section 4.1, it is necessary to flush the buffers on the PCI-SCI adapter between writing the data and the control information due to the possibility of reordering of writes by the underlying SCI network. This flush can cost up to five microseconds, because it involves reading status information on the adapter, thereby increasing latency considerably for small packets. The streams of the PCI-SCI adapter assures that sequential writes of up to 64 bytes to a 64 byte aligned remote memory area [6] are performed FIFO. We use this to optimize the latency of small packets as both data and control information can be written without the flush as long as they have a maximum length of 64 bytes. Instead of using the regular receive buffer, we have a specialized buffer for receiving such small packets. Each packet consists of a two byte header and up to 62 bytes of data payload. The header contains the packet length and an index used to deliver the packet at the correct place in the data-stream.

4.5 Implementation

SciStream is implemented under Linux 2.0.36 as a loadable kernel module and requires modifications to the Linux kernel as to integrate with the TCP/IP subsystem. SciStream interacts with the PCI-SCI adapter partly through Dolphin's Linux driver, partly through the SciOS [5] distributed shared memory system. SciStream does not use the advanced DSM capabilities of SciOS, but merely uses it for allocation and remote mapping of shared memory. As both SciOS and SciStream uses the regular interface to Dolphin's driver, they are able to co-exists with other interfaces to the SCI network. The limiting factor is the number of possible mappings of remote address space on the PCI bus, but this should not be a problem as SciStream uses only two concurrent mappings: one for the control data structures and one for data.

5 Performance

In this section we present latency and bandwidth microbenchmarks in order to give an overview of the performance of the current implementation of SciStream. We examine both a polling scheme where sender and receiver performs busy waiting and interrupt driven data exchange. Furthermore, we combine both of these with receive posting. Currently, we are unable to provide measurements for the page remapping facility. The hardware configuration used was two 450 MHz Pentium II based workstations with the Intel 440BX chipset and Dolphin's SCI-PCI revision D adapter. The workstations are connected to the same SCI ring, and they run Linux 2.0.36. The SciStream protocol is configured to use a receive buffer of a maximum of 256 KB, and receive posting is activated for packets larger than 2 KB.

The latency measurements were obtained by taking half of the average round trip time of 10,000 ping-pong packets. Figure 1 shows the latency of packets with sizes up to 256 bytes. The latency for 4 byte packet payload (resulting in a transfer of 6 bytes) is 8.4 microseconds when using polling. As the latency of SCI is 2 – 3 microseconds, we add 5 – 6 microseconds of overhead. Prior to the optimization for small packets the latency was close to 14 microseconds. The interrupt adds about 27 microseconds to the latency. Comparing the latency of packets smaller than 64 bytes with the larger packets one sees the effect of the 64 byte streaming performed by the PCI-SCI adapter. Each added 64 bytes only increases latency with a few microseconds. Figure 2 shows the latency of packets with size 4 KB and greater. In the interrupt driven case, receive posting is able to reduce latency by approximately 20%. The latency of the polling configuration grows at a smaller rate than that of the interrupt ones. This is because SciStream triggers a remote interrupt only when the full

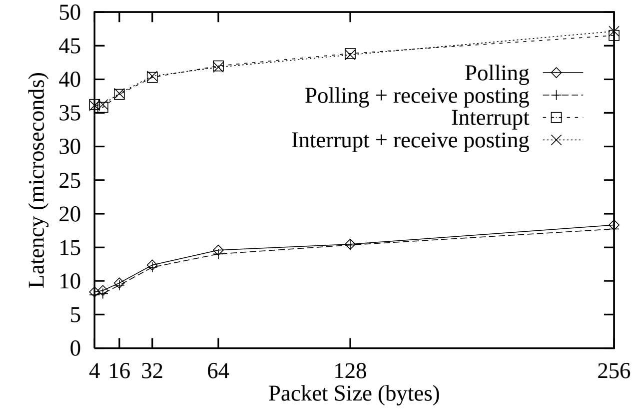

Figure 1: Latency for small packets

data payload has been transmitted, or when the receive buffer is full. When the receiver polls, it can start copying data as soon as the control information for a single page has been updated. With polling, receive posting increases overhead. The receiver is not able to perform the receive posting before the sender starts transmitting and thus the posting and invalidation of receive posted buffers add a small overhead.

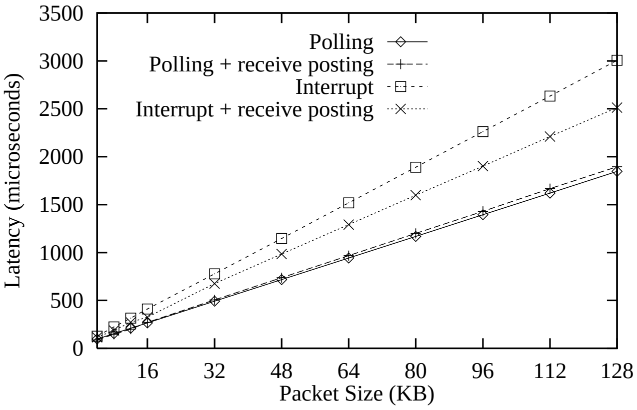

Figure 2: Latency for large packets

The bandwidth measurements were obtained by measuring the time it takes to transmit 10,000 packets. Figure 3 shows the bandwidth obtained with packets with sizes up to 4 KB, and we see that half of the maximum bandwidth of SciStream is achieved when using 1 KB packets. Figure 4 shows the bandwidth obtained with packets with size 4 KB and greater where the maximum throughput achieved is 72 Mbytes/s. The maximum raw copying bandwidth achieved on our current configuration is 84 Mbytes/s and thus system calls, flow control, and buffer flushing reduces the maximum bandwidth with 12 Mbytes/s. A negative effect of receive posting is also apparent in the measurements—it decreases the bandwidth with 1 – 2 Mbytes/s. Due to the use of programmed I/O, the sending side is the bottleneck and the receiver often blocks waiting for data and

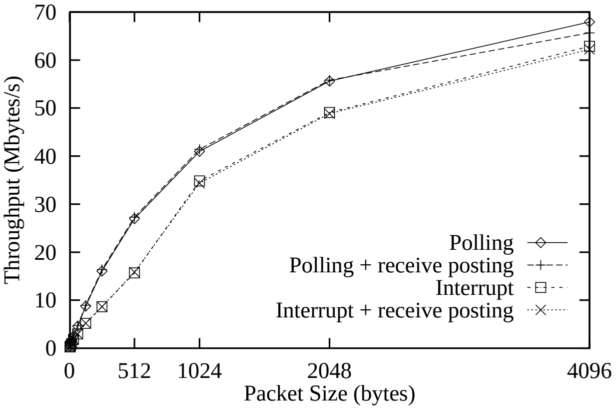

Figure 3: Bandwidth for small packets

receive posts a buffer. As the sender is constantly transmitting packets, it will always be transmitting the next packet when the receive posting happens, and all receive posting does is incurring overhead. This indicates that the receiver should stop using receive posting, if it repeatedly has to invalidate the receive posted buffer.

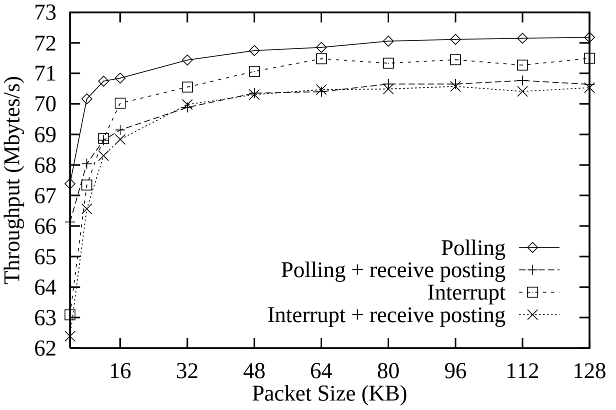

Figure 4: Bandwidth for large packets

6 Related Work

The introduction of new high speed networks such as Myrinet and SCI have resulted in several implementations of specialized stream protocols. Common for all of the implementations are that they are implemented at user level, and thus do not support page remapping.

Fast sockets [9] implements a stream protocol on top of active messages on Myrinet. Fast sockets introduced receive posting to reduce data copying but with the Myrinet interface, this does not require communication with the remote host, only with the local Myrinet interface.

The SHRIMP project has implemented a stream protocol [1] for their memory-mapped network interface using a separate protocol family. The functionality of the SHRIMP hardware resembles SCI closer than Myrinet.

They use a circular buffer for data transmission and do not employ techniques such as receive posting.

Two other specialized stream protocols for Dolphin's PCI-SCI adapter exist. Eberl et.al [3] compares several user-level communications libraries for SCI including active messages and a stream protocol, and SCILAN [8] is a stream protocol integrated with the Winsock interface. SCILAN uses the special mappings of the PCI-SCI adapter to reduce the number of interrupts and provides zero-copy reception by allowing senders to write directly into the receivers memory. Due to the user level implementation of the library, this can violate protection. Neither implementation support receive posting, nor the small packet optimization of SciStream.

Both the Basic Interface for Parallelism (BIP) [7] and PM [12] provide packet-based low level interfaces to Myrinet, and PM also exists in a version for Gigabit Ethernet (GigaE PM) [10]. Furthermore, PM implements a scalable flow control mechanism. Both interfaces provide very high bandwidth and low latency, and are intended to be used for message-passing parallel computing. They do, however, also provide an interface that allows the execution of the full TCP/IP protocol stack on top. SCIP [11] provides an interface to IP on top of SCI and thus also allows the execution of the full TCP/IP stack on top of SCI. Common for all these approaches is that running the full TCP/IP protocol stack on top of the low level interface provides full support for applications using TCP/IP, but they do increase latency and reduce bandwidth considerably compared to the raw low-level interfaces.

7 Conclusions

We have presented the design and implementation of a stream protocol for cluster-based SCI networks. The protocol has been integrated with the TCP/IP subsystem of a general operating system and is able to provide access to the high bandwidth and low latency of SCI networks for unmodified applications using TCP. We employ several optimizations aimed at providing lower latency for client-server relationships. For small packets, we decrease latency by exploiting features specific for the PCI-SCI adapter. The latency of large packets can be reduced with up to 20 % by using the receive posting techniques modified for SCI networks. To handle the cases where receive posting cannot avoid data copying, we are currently investigating page remapping optimizations. We find that it is possible to expose the raw performance of the SCI network to user level processes without compromising protection.

Acknowledgements

Kåre Løchsen and Hugo Kohmann from Dolphin Interconnect Solutions (Norway) willingly granted us access to the source codes for the PCI-SCI adapter and answered all of our questions. Roger Butenuth from Universität Paderborn (Germany) kindly gave us access to his Linux port of the PCI-SCI driver. The SIRAC group at INRIA, especially Emmanuel Cecchet and Xavier Rousset de Pina, provided us with valuable support for the SciOS prototype.

References

[1] S. N. Damianakis, C. Dubnicki, and E. W. Felten. Stream sockets on SHRIMP. *Lecture Notes in Computer Science*, 1199, 1997.

[2] Dolphin Interconnect Solutions. PCI-SCI cluster adapter specification, May 1996. Version 1.2. See also `http://www.dolphinics.no`.

[3] Michael Eberl, Hermann Hellwagner, Wolfgang Karl, Markus Leberecht, and Josef Weidendorfer. Fast communication libraries on an SCI cluster. In *Proceedings of SCI Europe 98*, pages 165–175, Bordeaux (France), September 1998.

[4] IEEE. *IEEE Standard for Scalable Coherent Interface (SCI)*. 1992. Standard 1596.

[5] Povl T. Koch and Xavier Rousset de Pina. SciOS: Flexible operating system support for SCI clusters. In *Proceedings of the 4th International Euro-Par Conference*, number 1470 in Lecture Notes in Computer Science, pages 601–609, Southampton (UK), September 1998.

[6] Knut Omang. Synchronization support in I/O adapter based SCI clusters. In *Proceedings of Workshop on Communication and Architectural Support for Network-based Parallel Computing (CANPC'97), Lecture Notes in Computer Science no.1199*, pages 158–172, feb 1997.

[7] L. Prylli and B. Tourancheau. BIP: A new protocol designed for high performance networking on myrinet. *Lecture Notes in Computer Science*, 1388, 1998.

[8] Stein Jørgen Ryan and Haakon Bryhni. SCI for local area networks. Technical Report 256, Department of Informatics, University of Oslo, 1998.

[9] Steven H. Rodrigues, Thomas E. Anderson, and David E. Culler. High-Performance Local Area Communication with Fast Sockets. In *Proceedings of the USENIX 1997 Annual Technical Conference*, 1997.

[10] Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa. High performance communication using a gigabit ethernet. Technical Report TR-98003, Real World Computing Partership, 1998.

[11] Hüseyin Taşkin, Roger Butenuth, and Hans-Ulrich Heiss. SCI for TCP/IP with Linux. In *Proceedings of SCI Europe 98*, pages 155–157, Bordeaux (France), September 1998.

[12] Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. PM: An operating system coordinated high performance communication library. In *High-Performance Computing and Networking '97, Vol. 1225 of Lecture Notes in Computer Science*, pages 708–717, April 1997.