

A Cluster Testbed for SCI-based Parallel Processing

Alan D. George, Robert W. Todd, and Warren Rosen*

*High-performance Computing and Simulation (HCS) Research Laboratory
Electrical Engineering Department, FAMU-FSU College of Engineering
Florida State University and Florida A&M University*

Abstract

A critical factor in the design of any parallel processing computer system is the interconnection network between processors. Latency, throughput and other network factors weigh heavily on the performance of any parallel program. In this paper, the average latency and sustained throughput for a ten-node, SPARC-based SCI cluster is tested and measured based on variations in message size, cluster or ring size, and transfer method. These results are compared with corresponding TCP/IP-based Ethernet values in order to help gauge the potential of COTS SCI clusters for supporting parallel processing environments and applications.

1. Introduction

Parallel processing has emerged as a key enabling technology for a wide variety of current and future applications spanning both the commercial and military sectors, however these applications continue to rely heavily on the speed of the interconnection system used for processor communication. For years, advancements in processor technology have consistently outpaced those in interconnect technology in terms of performance, cost, and availability. However, new interconnect technologies such as SCI promise to close the gap and support the development of new, more powerful, and more effective parallel processing systems.

Historically, two basic approaches have been used: custom, high-performance interconnects in high-cost supercomputer and massively parallel processor (MPP) systems; and commercial, off-the-shelf (COTS), standardized computer networks for interconnecting processors in the form of a multicomputer system. The first approach is expensive, largely proprietary, and lacking in portability; the second supports only coarse-grain parallel algorithms due to poor latency and throughput and lack of support for a shared-memory style of programming.

Recently, a new computer interconnect has emerged which has the potential to revolutionize the way in which parallel processing systems are designed and implemented. SCI is unique among standardized interconnects and networks in that it supports cache-

coherent, shared memory between physically distributed processors and memory units in a fashion scalable to tens of thousands of processors, while supporting a variety of topologies and specifying link data rates of eight gigabits per second and latencies expected to drop below a microsecond [SCI93]. By combining these and other technical advantages with the logistic and cost advantages of international standardization and COTS availability, it may be that SCI is at present uniquely qualified as a parallel processing interconnect technology [GUST95].

In the next section an overview of the cluster testbed is provided. This is followed by the presentation and discussion of a series of latency and throughput tests conducted on the testbed. These results provide an indication of the range of message sizes and cluster sizes where the SCI/Sbus-1 interconnect is superior and others where it is inferior as compared to the conventional interconnect for workstation clusters (i.e. 10-Mbps Ethernet). Next we overview several parallel programming environment tools which are likely to be available in the near future for parallel processing on SCI-based clusters. Finally, a brief set of conclusions and topics for future research are provided.

2. Testbed Configuration

For the purpose of studying the advantages and disadvantages of SCI-based parallel processing, a cluster testbed has been procured, installed, and made operational at the *High-performance Computing and Simulation (HCS) Research Laboratory*. As illustrated in Figure 1, the cluster configuration is based on ten SPARCstation-5/85 clones (i.e. with 85-MHz MicroSPARC-II processors) connected by Dolphin SCI/Sbus-1 adapters in a simple ring topology operating at a link data rate of 1-Gbps.

The workstations operate under SunOS 4.1.3_U1 and communicate through TCP/IP over a 10-Mbps thinwire Ethernet LAN for routine UNIX traffic. Communication with the SCI ring is accomplished with standard UNIX I/O functions for message passing and proprietary functions from the vendor for shared-memory creation, mapping, and destruction.

* Dr. Rosen is with the Naval Air Warfare Center, Aircraft Division, Warminster, PA

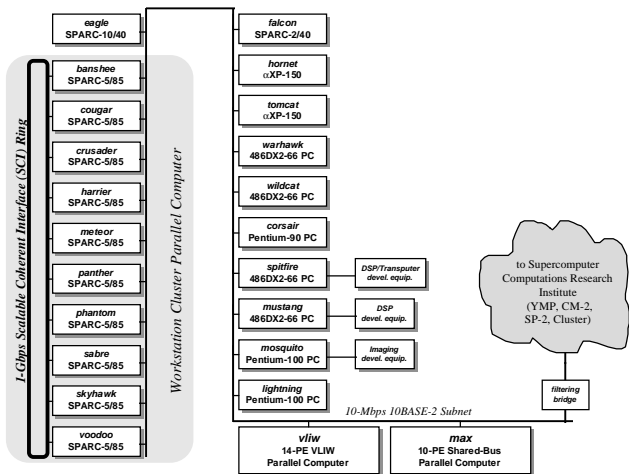


Figure 1. HCS Lab Computing Facilities

The functional components of the SCI/Sbus-1 adapter cards are shown in Figure 2 [DOLP95][ALNE93]. These adapters currently support both message-passing and limited, non-coherent, shared-memory interprocessor communication techniques which when mature will be available as API (application program interface) function calls from any high-level-language or assembly-language program.

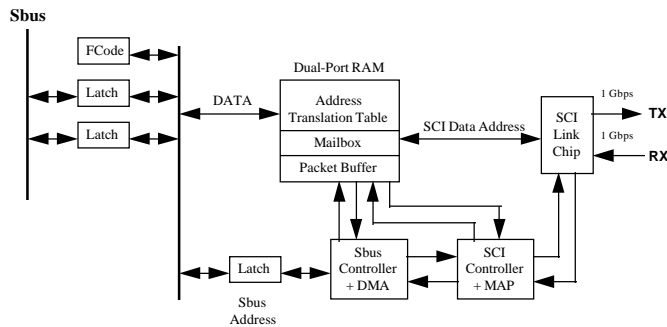


Figure 2. SCI/Sbus-1 Adapter (courtesy of Dolphin Interconnect Inc.)

Virtual shared-memory on a physically distributed memory system is handled by the Address Translation Table where 32-bit Sbus addresses are converted to 64-bit SCI addresses. The most significant 16 bits of the SCI address are used to select between 65536 distinct devices. To map this SCI “address” space into a user address space the *mmap()* function is called which makes a copy of the buffer allocated in the kernel space and places it in the user application’s virtual memory segment. This allocated buffer is created via an *ioctl()* command and is mapped via the *mmap()* command. The DMA engine that underlies this shared-memory system provides the ability to transparently move data from local memory on one station to local memory on another. Shared-memory transfers are accomplished, at the most fundamental level, by simple assignment

operations where a pointer to the shared-memory segment is assigned a value which then is sent, via the DMA engine, to all other local copies of the shared memory buffer on the ring. By doing this, another node on the ring has the ability to “see” that value.

The message-passing API consists of normal I/O command structures such as *read()*, *write()*, and *ioctl()*. These commands normally operate on simple file descriptors which act as sockets to the Sbus itself and thus are created via *open()*. These transactions proceed into message FIFOs where the link controller chip decides how to break apart the message and push it across the link. For optimal transfers however, the write buffer should be aligned to a 64-byte boundary before passing it to the Sbus. This allows the SCI card to take direct advantage of fast 64-byte transfers without the need for breaking apart a message. Sbus commands are mapped to SCI commands and back again using the simple table shown in Table 1.

Table 1. Sbus to SCI Mapping [DOLP95]

Sbus cmd	Sbus size	SCI Request	SCI Response
<i>read()</i>	$2^0 - 2^3$ bytes	<i>read_sb()</i>	<i>response_16</i>
<i>write()</i>	$2^0 - 2^3$ bytes	<i>write_sb()</i>	<i>response_00</i>

3. Throughput and Latency Tests

In order to begin to gauge the potential performance benefits of the SCI interconnect in general and in a workstation cluster environment in particular, a number of basic benchmarking programs have been developed and their results measured and collected. These benchmarks consist of various test programs which call SCI/Sbus driver functions associated with either shared-memory or message-passing interprocessor communication, and include both single-source, single-destination tests and multiple-source, multiple-destination tests with varying message sizes. The results of these basic experiments are provided in the form of both latency and throughput measurements.

The approach taken to measure the amount of throughput and latency is based on extensions to basic ping-pong tests of 4-byte words. However, in the cases presented here message sizes, communication methods, and ring sizes were varied to produce a suite of benchmark test results for the SCI/Sbus-1 implementation of the SCI standard.

For throughput, both message-passing (MP) and shared-memory (SM) programming models were used. MP programs perform a specified number of simple *read()* and *write()* operations to an SCI channel with a buffer of the desired message size. The buffer was aligned to a 64-byte boundary for optimal efficiency with the DMA operations inherent to the implementation. SM programs create a shared-memory segment of the desired message size plus one byte to

accommodate a flagging mechanism for acknowledged receipt of a message. The shared-memory segment physically exists on the destination machine thus for very small transfers the one-byte flag provides a minimal skew to the throughput measured. The source machine then writes a number of bytes to the shared-memory segment and waits for the acknowledge flag byte to be set before it can perform the next iteration. The time needed to transfer a fixed amount of bytes to the destination machine divided into this amount is the throughput of the communication network. Figures 3-4 summarize the results of our sustained throughput tests for a single sender to a single receiver on rings varying from two to ten nodes and compared to basic TCP/IP over Ethernet in terms of the message-passing approach and the shared-memory approach respectively. In the case of message passing, the throughput of SCI/Sbus-1 peaks at approximately 32-34 Mbps for small and large rings and begins exceeding Ethernet once the message sizes reach 1K bytes. By contrast, the throughput of shared-memory accesses cannot compete with Ethernet except for very small message sizes. Overall, the SCI/Sbus-1 nodes are each capable of sustained throughput of almost six times that of Ethernet.

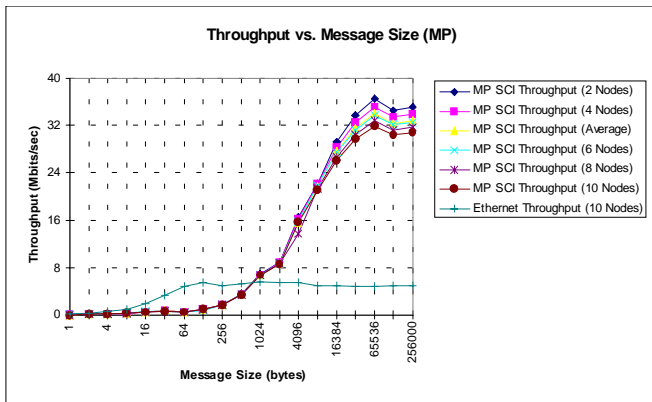


Figure 3. MP Sustained Throughput for Single-Source, Single-Destination

In the case of latency, similar methods were employed to create a complementary set of programs to the throughput set. For the message-passing model, again *read()* and *write()* functions were performed a desired number of times using different buffer sizes corresponding to the experimental message sizes. However, in this case the message is sent to the destination, and the full message is immediately returned upon receipt. The total round trip time of the specified message size is the two-way latency.

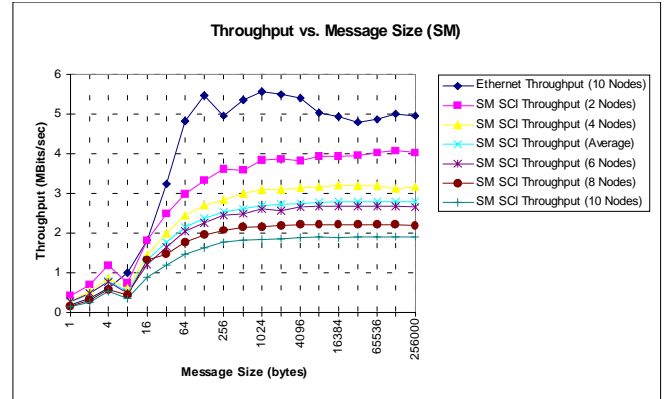


Figure 4. SM Sustained Throughput for Single-Source, Single-Destination

Figures 5-6 summarize the results of our two-way or ping-pong latency tests for a single sender to a single receiver on rings varying from two to ten nodes and compared to basic TCP/IP over Ethernet in terms of the message-passing approach and the shared-memory approach respectively. In the case of shared memory, the two-way latency of SCI/Sbus-1 starts at approximately 24-30 microseconds with one or four bytes transfers over small and large rings and begins to suffer as compared to Ethernet once the message sizes go beyond 256 bytes. By contrast, the latency of message-passing accesses cannot compete with Ethernet except for very small message sizes.

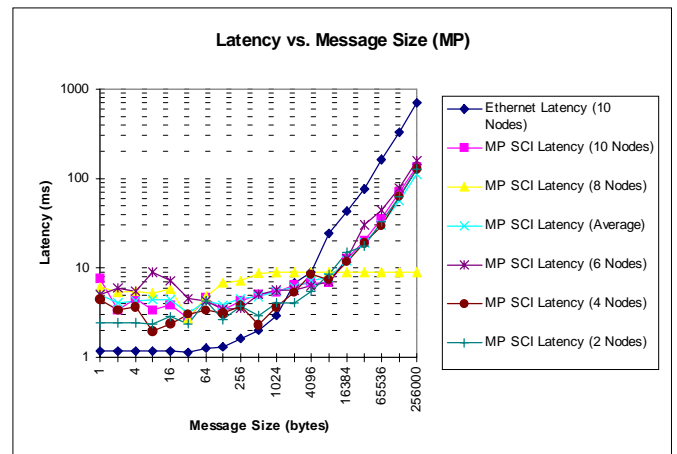


Figure 5. MP Two-way Latency for Single-Source, Single-Destination

Thus, as might be anticipated, the SCI ring shows its most promising behavior in terms of latency for shared-memory access with small messages and in terms of sustained throughput for message-passing access with large messages.

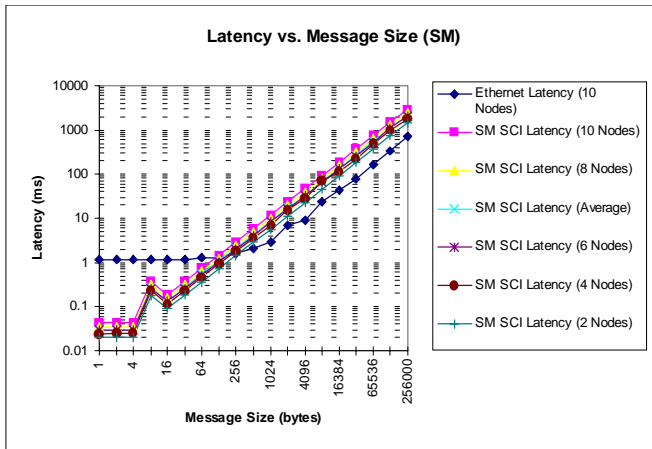


Figure 6. SM Two-way Latency for Single-Source, Single-Destination

While the maximum attainable sustained throughput and the minimum attainable latency were not quite as good as those reported by the adapter and workstation vendors, these differences may perhaps be attributed to the use of SPARCstation-5 (SS5) machines in our cluster versus SPARCstation-20 (SS20) machines in other tests. While the SS5s are computationally as fast or faster than some SS20 models, their Sbus interfaces are slower and offer less functionality. Further investigations are being made to better quantify these Sbus interface differences with respect to the relative influences of differences in Sbus speeds, memory-mapped I/O access methods, caching, etc.

In addition to these single-source, single-destination experiments, a number of tests were also conducted using multiple-source, multiple-destination cases to again measure sustained throughput but this time with respect to the ring itself. The test cases considered multiple senders up to all ten nodes and varying traffic patterns including such (source,destination) ordered pairs of SCI node numbers as $\{(0,1), (2,3), (4,5), (6,7), (8,9)\}$, $\{(1,0), (3,2), (5,4), (7,6), (9,8)\}$, $\{(0,5), (1,6), (2,7), (3,8), (4,9)\}$, $\{(0,1), (1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,0)\}$, etc. as well as pseudorandom patterns. In all cases tested, the throughput results proved to be virtually linear with respect to the individual sustained throughput rates times the number of senders. All measurements taken thus far show that the bottleneck associated with the SCI/Sbus interface dwarfs any potential congestion that can be injected into the network for clusters of ten or less SS5 nodes. Of course, if the number of nodes were increased to the ratio of peak throughput (e.g. 1.0 Gbits/second per link) to the per-node sustained throughput (e.g. 36 Mbits/second), the results would be quite different. However, based on the measurements made thus far this would mean perhaps upwards of twenty to thirty SS5

nodes before a significant negative impact is experienced.

4. Parallel Programming

As the measurements in the last section indicate, even with these first-generation SCI/Sbus adapters the latency and throughput afforded by an SCI-based cluster for certain message sizes are far superior than that provided by the conventional interconnect for workstation clusters. By taking advantage of the message-passing or limited shared-memory modes of the SCI adapters it is possible to design any number of parallel programs.

While "scratch" test programs can be hand-coded with calls to vendor-supplied driver functions and message-passing and shared-memory API functions as they mature (current API and driver functions are still in their infancy), to effectively harness the parallel processing potential of an SCI-based cluster in a manner that is compatible with the parallel processing community at large it is important that parallel programming paradigms be employed and new tools be ported or developed. While the hardware aspects of parallel processing are certainly challenging and new developments continue to be needed, it is a widely accepted fact that the software side of parallel processing lags behind the hardware side and most of the difficulties and costs associated with making parallel processing an effective mechanism for mainstream computing lie with the software. The development of parallel algorithms and applications that efficiently map to the underlying architecture in a manner that is portable is extremely difficult to accomplish in a manner with any degree of breadth. In order for an SCI-based cluster to be a legitimate architecture for parallel processing, such parallel programming tools much be provided.

For instance, one of the most widely used parallel programming tools for workstation clusters is the Parallel Virtual Machine (PVM) software developed and distributed for free by the Oak Ridge National Laboratory [SUND90]. This tool provides the programmer with a set of message-passing functions used with a high-level language program and has been ported to many platforms including machines on which these functions are physically implemented via shared-memory access as well as distributed-memory machines such as workstation clusters. Another similar model of parallel computation is the Message Passing Interface (MPI) specification developed by the MPI Forum, and it too is available for a growing number of distributed- and shared-memory platforms [MPIF93]. Unlike PVM and MPI which require explicit parallelism and parallelization by the user, a very different approach to parallel program design is afforded by High Performance Fortran (HPF) [LOVE93]. HPF follows a data-parallel

model of computation primarily aimed toward algorithms and applications whose domains can be decomposed such that the same operation can be applied to some or all elements. While not a true parallelizing compiler, array data structures and operations coupled with data distribution and other compiler directives provide the software designer with much more implicit support for parallel programming of data-parallel problems. Somewhat similar is the Parallel Applications Management System (PAMS) which provides preprocessor directives for C and Fortran compilers supporting constructs such as parallel DO loops while alleviating the user from involvement with interprocessor communication [KARP93]. While there are literally dozens of parallel programming environments like these, ranging from implicit data-parallel languages to explicit message-passing ones, these four tools (i.e. PVM, MPI, HPF, and PAMS) are particularly interesting in the short term as several vendors are developing SCI/Sbus versions. Several of these tools are currently in Beta test and are being evaluated in the HCS lab.

5. Conclusions and Future Research

A number of basic latency and throughput tests have been conducted and measurements collected. These results have helped to better understand the basic communications behavior when using SCI/Sbus-1 adapters as a workstation cluster interconnect. Even with these first-generation SCI/Sbus adapters the latency and throughput afforded by an SCI-based cluster for certain message sizes are far superior than that provided by the conventional interconnect for workstation clusters (i.e. TCP/IP running over 10-Mbps Ethernet). We believe that further improvements can be achieved for throughput and perhaps even latency with additional optimizations to the API and drivers, and these options are being investigated.

The development of the SCI-based cluster testbed has just begun. The potential for SCI as an interconnect for parallel processing systems is promising, both in terms of commercial systems including MPPs and workstation clusters and military systems, but much work remains before it may achieve widespread acceptance. Current hardware and software availability is limited, and while the performance of SCI-based clusters surpasses that of conventional Ethernet-based clusters, limitations of first-generation nodechips and especially the Sbus interface bottleneck must still be overcome before the gigabyte-per-second throughput and submicrosecond latencies can be achieved. Software tool development must also continue, and the pending release of several parallel and distributed programming tools is encouraging.

A number of future research tasks remain in order to better understand the issues associated with SCI as a parallel processing interconnect. From a theoretical standpoint, work has been proposed for modeling and simulation of SCI parallel processing architectures and algorithms. The study of algorithm decomposition and mapping to various SCI-based architectures and performance analyses based on these efforts may hold the key to better understanding and exploiting the potential of SCI parallel processing both in terms of theoretical peak performance and practical realizable performance. Continuing hardware upgrades are anticipated for the cluster testbed in the coming semesters, including the addition of packet-switched and multiservice switches supporting several topologies and the enhancement of SCI/Sbus adapters in the short term and use of processor bus adapters thereafter. Beta tests with several parallel programming tools ported to SCI will continue, and as performance stabilizes a number of parallel performance experiments are planned comparing SCI-based clusters with several different topologies to other clusters based on TCP/IP and Ethernet, FDDI, Fibre Channel, ATM, and proprietary interconnects such as the enhanced Omega network in the SP-2. It is our goal to leverage hardware and software developments taking place at various sites, collaborate when possible, and construct or develop critical elements as needed.

Acknowledgements

We gratefully acknowledge the support of our sponsors at the National Security Agency in Fort Meade, Maryland, the Naval Air Warfare Center, Aircraft Division, in Warminster, Pennsylvania, and the Office of Naval Research in Washington D.C. We also wish to thank Bjorn Dag Johnsen and Knut Alnes at Dolphin Interconnect Solutions in Oslo, Norway for their technical support, and M. Miars, D. Zirpoli, W. Phipps, M. Sarwar, and D. Collins in the HCS lab for their assistance in collecting measurements.

References

- [ALNE93] Alnes, K., "Enabling Products for Cluster Computing using SCI," *Proceedings of the First International Workshop on SCI-based High-Performance Low-Cost Computing*, pp. 58-64, August, 1994.
- [DOLP95] Dolphin Inc., "1 Gbit/sec SBus-SCI Cluster Adapter Card", White Paper, Dolphin Interconnect Solutions, March 1995.
- [GUST95] Gustavson, D.B. and Q. Li, "Local-Area MultiProcessor: the Scalable Coherent Interface," *Proceedings of the Second International Workshop on SCI-based High-*

Performance Low-Cost Computing, pp. 131-154, March, 1995.

- [KARP93] Karpoff, W. and B. Lake, "PARDO - a deterministic, scalable programming paradigm for distributed memory parallel computers and workstation clusters," *Proceedings of Supercomputing 1993*, IEEE Computer Society Press, 1993.
- [LOVE93] Loveman, D., "High Performance Fortran," *IEEE Parallel and Distributed Technology*, Vol. 1, No. 1, pp. 25-42, 1993.
- [MPIF93] Message Passing Interface Forum, "MPI: A Message Passing Interface," *Proceedings of Supercomputing 1993*, IEEE Computer Society Press, pp. 878-883, 1993.
- [SCI93] *Scalable Coherent Interface*, ANSI/IEEE Standard 1596-1992, IEEE Service Center, Piscataway, New Jersey, 1993.
- [SUND90] Sunderam, V.S., "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice & Experience*, Vol. 2, No. 4, pp. 315-339, December 1990.