

GASNet Core API on Scalable Coherent Interface: Design and Realization

Hung-Hsun Su, Burt Gordon, Sarp Oral, Alan D. George
{ su, gordon, oral, george }@hcs.ufl.edu

*High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32611*

Keywords – *Unified Parallel C (UPC), Global Address Space Networking (GASNet), Scalable Coherent Interface (SCI), Active Message (AM).*

Abstract

UPC is a promising programming model for shared-memory parallel computing on shared- and distributed-memory system architectures. Berkeley UPC, which utilizes the GASNet communication system, is one promising system aimed to develop a portable, high-performance implementation of UPC for large-scale clusters. Scalable Coherent Interface (SCI) is known for its ability to provide very low latency transfers and direct support of a global shared-memory address space. High-performance parallel clusters built with SCI appear as a promising platform for large-scale UPC applications. This paper introduces the design of the GASNet SCI Conduit Core API. Data structures and algorithms needed to support Active Message (Core API) on top of SCI SISI API are described and performance results of preliminary tests are included.

1 Introduction

Many scientific as well as commercial endeavors rely on the ability to solve complex problems in a quick and efficient manner. One of the dominant solutions to this problem has been the emergence of parallel computing. To supplement the architectural improvements in this area, parallel programming models have emerged to give programmers alternate ways to approach solving complex and computationally intensive problems. Such models include message passing, shared memory, and distributed shared memory.

While message passing and shared memory are the two most popular ways to implement parallel programs, distributed shared memory is quickly gaining momentum. One of the reasons for this development is the emergence of Unified Parallel C (UPC). UPC is a parallel extension to the ANSI C standard. It allows programmers the ability to create parallel programs across several parallel

architectures while still maintaining a familiar C style structure. This approach allows a smaller learning curve for people with C experience to begin creating parallel programs.

One of the recent developments in UPC is the interest in providing a means for executing UPC over clusters of computers. The most promising tool available now for implementing UPC on clusters is the Berkeley UPC runtime system developed by U.C. Berkeley and LBNL. An underlying key to this system is the GASNet communication layer. GASNet allows a standard application interface to be implemented over a wide variety of standard and high-performance networks.

High-performance networks are critical for creating high-performance clusters of computers by acting as the interconnecting fabric in System-Area Networks (SAN). A SAN-based system is a group of computers interconnected by a high-performance interconnect. The Scalable Coherent Interface (SCI) is one such high-performance interconnect.

In this study we design and prepare a GASNet conduit to be executed over the SCI network. To ensure that the conduit provides for high performance, we run benchmarks on GASNet being implemented on an MPI conduit on SCI and on our newly developed conduit. Comparisons are made between the two conduits and the raw performance of SCI using the SISI API in order to show strengths and weaknesses.

The next section of this paper briefly describes the architecture of Dolphin's SCI interconnect with its SISI API and Berkeley's GASNet communication layer. In Section 3, we discuss related research. Section 4 discusses the design of GASNet/SCI conduit. Section 5 presents the preliminary performance results. Section 6 presents conclusions and future plans.

2 Component Overview

Understanding the underlying network is essential to creating an efficient conduit for the GASNet system. In the following subsections we

present an overview of the SCI network and a brief introduction of the GASNet communication system.

2.1 SCI

The Scalable Coherent Interface (SCI) is an ANSI/ISO/IEEE standard (1596-1992) that describes a packet-based protocol [2]. SCI was developed as an attempt to address the problems associated with buses for use with many processors. SCI offers many clear advantages for the unique nature of parallel computing demands. Perhaps the most significant of these advantages is its low-latency performance, typically on the order of single-digit microseconds. SCI also offers a link data rate of 5.3 Gb/s in current systems. Another advantage in using SCI is that, unlike competing systems, SCI offers support for both the shared-memory and message-passing paradigms. All transactions are seen as shared-memory operations through the SCI address space.

SCI uses 64 bits in its address translation. The most significant 16 bits are used to specify the node and the remaining 48 bits are used for addresses within each node. By this scheme, the SCI environment can support up to 65,536 nodes with 256TB of addressable space [2]. Through API function calls, one can map local memory segments to and import memory from the SCI address space.

The SISI API specification is a standard set of API calls used to execute transactions in SCI [4]. The SISI API allows users to directly access the hardware and control its behavior. In order to communicate between two nodes, each node must set aside a portion of its physical memory for use by the SCI network. A local node will then create a memory segment, prepare it, map it locally and set it available for other nodes to connect to it and map it into their own address space [4]. When remote memory is mapped to the local address space, only the size of a pointer will be used. Any transactions from the base address will be controlled by the SCI driver and written to the appropriate place.

The SCI conduit design and analyses featured in this study are based on the Dolphin D339 NIC. We use both the Dolphin SCI driver version 1.31, which includes an implementation of version 2.11 of the SISI API, and the Scali SCI driver found in SSP 3.1, which includes ScaMPI, a high-performance implementation of MPI for SCI. Dolphin SCI hardware is created in such a way that remote writes are approximately 10 times faster than remote reads. This disparity is due to the inability of streamlining reads through the memory PCI bridges (information obtained from Dolphin support). So to obtain the best performance, remote reads need to be avoided and replaced by either local reads or remote writes.

2.2 GASNet

Global Addressing Space Networking (GASNet) [5], developed at Berkeley/LBNL, is a language-independent, low-level networking layer that provides high-performance communication primitives that are network-independent and aimed at supporting parallel languages such as UPC [7] (Unified Parallel C) and Titanium (Parallel dialect of Java). The system is divided into 2 layers, GASNet Core API and GASNet Extended API. The Core API is a narrow interface based on the Active Message paradigm and Firehose [6] algorithm (memory registration strategy) specific to each network. Currently, GASNet supports execution on MPI, Myrinet, Quadrics and IBM LAPI [12]. The Extended API is a network-independent interface that provides medium- and high-level operations on remote memory and collective operations (Figure 1).

3 Related Research

GASNet is being developed at Berkeley/LBNL and was first released on 1/29/2003 with latest release (v 1.3) on 4/17/2003. As it is still very new, little in the way of published papers is available. As the time this paper is written, the most detailed source of information on GASNet is the UC Berkeley website [12]. SCI shared memory as well as SAN-based interconnect is a fairly mature research area; among the more notable studies related to this project are [9], [10], and [11].

As the GASNet Core API is basically Active Messages (AM) over SCI, Ibel's paper [10] discusses several possible ways to execute AM over SCI and is instrumental in guiding our design. One of the most important contributions is the suggestion of using several different buffers, or segments, to deal with messages from different sources. This approach will be explained further in the paper. Additionally, the several available GASNet conduits (see GASNet source code) are important and practical design and coding guides.

4 Design of Conduit – Core API

To enable remote shared-memory operations, the SISI API requires that each receiver exports (prepare a part of its local memory to be globally accessible) one or more global memory region (SCI segments) and each sender imports (register the remote memory region) one or all of these global memory regions into its local virtual address space. Once imported, a sender can read and write the receiver's global memory region directly as if it is local. SISI automatically converts any access to the virtual address space into network transfers to the appropriate machine's physical memory.

4.1 Data Structures

As noted previously, design of the GASNet SCI conduit Core API centers around the need to support Active Messages on SCI. The challenge is to enable message-passing operations (send/receive) using shared-memory operations (read/write). To support this, various data structures are created and the usage of each is explained in the following subsections. In addition, it has been determined that remote read operations are 10 times slower than remote write operations. As a result, our design utilizes remote writes and local reads exclusively and some of the data structures are used to achieve this end.

4.1.1 Handler Table

This structure is an array of 256 (as per GASNet spec) `void*` pointers representing function pointers (Figure 2A). It stores all internal and user-defined handlers. This table is needed to invoke the correct handler for incoming active messages (index of array = handler ID).

4.1.2 Local Segment Pointers (LSP), Remote Segment Pointers (RSP)

Segment pointers are arrays holding the starting memory addresses (Figure 2B) of all exported (local), and imported (remote) SCI segments. LSP is used to process incoming messages, whereas RSP is used to transfer data to remote nodes.

4.1.3 Command Message (AM Header)

This structure is used to hold all information needed for a single active message (message header, Figure 2C, 2 types). *Handler* stores the handler ID, *Message Type* is either a request or a reply, *AM type* marks if the message is a short, medium or long AM, *Payload Pointer* and *Payload Size* are used for medium (the medium payload pointer is implicit as its address is always at an offset from the header) and long Active Messages, *Num Args* and *Args N* store the number of arguments and the actual arguments that will be passed to the handler. Finally, *next* and *token* pointers are added on the receiver side to speed up polling (*next* links the message together while *token* stores the necessary information). Each outgoing message is first packaged with this command message structure and then copied into receiver's memory location. The receiver then (with addition of *next* and *token*) obtains all necessary information by accessing the sub-fields of this structure. This level of indirection supports message passing using shared-memory operations.

4.1.4 Message Location Status

The Message Location Status is an array of flags used by the local node to keep track of the status of all outgoing command messages. Flag for a particular node and message number is set to 1 when a message is transferred from local node to the destination node (message number < 2 times the number of Request-Reply pairs). Flag is reset to 0 only when a reply is received from the destination node indicating that the destination node handled the message. The local node first checks for a free slot, by checking if any of the flags are 0, before sending a new request and reply pair.

4.1.5 Message Ready Flags (MRF) / Message Exist Flag (MEF)

The MRF is an array of bytes where each sender has a particular spot on a receiver in which to write when a data transfer occurs. The MEF is a single spot on each receiver that is globally accessible by all nodes. This flag is used to notify the receiver if any node has sent a data transfer. A specific MRF is set if there is a new message from a particular sender (value indicates the message type, SHORT = 10, MEDIUM = 11, LONG = 12, CONTROL = 13) and the MEF flag is set if there is any new message from any sender. The sender writes to these flags after completely transferring a message to signal the receiver of the existence of a new message to be processed. The MRF must be written after completion of message transfer and MEF must not be written before MRF is set to avoid starvation. These flags add additional overhead to message reception and handling, however most of the overhead can be transparent, as SCI allows multiple writes to be posted at once and the data sent is only one byte.

4.1.6 Work Queue

The Work Queue is a FIFO linked list of memory addresses (Figure 2D) where each address represents the starting location of a pending, incoming active message. Together with MRF and MEF flags, these structures are used to achieve constant message poll time. Figure 3 details the steps each receiver follows when polling a new message.

4.2 Memory Management

This section provides the memory management schema for the SCI Conduit Core API that uses the data structures described above. The simplest, most direct design is to have each node export one

contiguous region and every node import every region from every other node (Figure 4). To send a message, the sender will need to find a free portion within receiver's SCI segment that can hold the new message (thus avoiding a race condition) before sending a message. This process generally involves the use of locks and introduces extra network transactions needed for message transfer. Furthermore, scalability of the system becomes a concern if the global memory region that each node exports is large. Finally, message polling becomes a concern since there is no good strategy to signal the arrival of a new message and where it is located.

A similar scheme was used in Ibel's [10] first attempt at supporting Active Messages on SCI. He soon realized that this approach had many drawbacks and suggested improvements to this design. In order to obtain better performance, several improvements over the simplest solution were used based on Ibel's findings. These approaches have been modified from our own experiences and are discussed below.

On each node, our design divides the exported SCI segment into command segments (N total) and a long AM payload segment. Each command segment is usable only by one sender (dedicated channel, Figure 5) whereas all senders operate on the same long AM payload segment. The command segment is large enough to accommodate 4 medium messages (2 requests, 2 replies) with payload size up to 944bytes (1024 – header size). At system initialization (Figure 6), each node exports all segments but only imports one command segment from all other nodes (its dedicated channel into that node). The long payload region is imported during system execution phase in order to improve the scalability (sacrificing performance, a design influenced by the Berkeley GASNet team).

Experimental results show that SCI shared memory operations (PIO) are better for smaller messages, whereas SCI DMA operations are more suited for larger messages (Figure 7). To take advantage of this tradeoff, our design limits the size of AM medium message to < 1KB so that short and medium messages can be transferred utilizing PIO. Since the size of a long AM payload could be of any size and is assumed to be generally large, we use DMA to transfer the long AM payload in addition to transfer of long AM header using PIO (Figure 8).

With this setup, our design is tuned to use the best possible transfer mode (PIO/DMA) with respect to the size of the message. Furthermore, race conditions among senders are prevented without the use of locks (fewer network transactions, no remote read). Using these mechanisms with the MEF, MRF and work queue, message polling becomes simple and efficient.

4.3 Bootstrapping

In SCI, a node becomes a part of a system by creating (which can only be called locally) a portion of its local memory as shared region and exporting it to every other node in the system. Once this step is complete, all other nodes can communicate with this node by placing messages in this memory region.

To start up the system, a script is run at node 0 which will then remote shell into each participating node to setup the environment. Setup involves declaring the shared memory region, importing the remote shared memory region, local handler table, lock table, message queue generation and initializing various environmental variables (fig 9).

Machine/SCI node mapping is read from a configuration file. Using this information, the system will determine which nodes to use given a user-defined system size. Optionally, a user-defined job file (list of machine names) could be used to specify the nodes the current application will use (instead of going down the configuration list to find the first X number of nodes, the job file can force the application to use the specified machines). Given this information, the system can determine the complete mapping between machine, GASNet node ID, and SCI node ID for the current application.

5 Preliminary Results

A generic GASNet conduit already exists for any MPI software system. Scali produces ScaMPI, the most widely used implementation of MPI for SCI. In this section we present a preliminary performance comparison of basic GASNet Active Message functions over our new SISCO-based conduit versus the MPI conduit over ScaMPI.

5.1 Experimental Setup

This section introduces the environment and testing procedures used in obtaining performance numbers for each of the software environments. The experiment setup, as well as the design options, are described in some detail.

5.1.1 Testbed

The system used in this study is a set of 16 nodes each with dual 1GHz Pentium-III CPUs with 256KB L2 cache, 256MB PC133 SDRAM, 133MHz system bus, with 64-bit, 66MHz PCI slots. Each node uses Linux Red Hat 8.0 with kernel 2.4.9-14 and has gcc version 3.2. The SCI clusters are set up as two 4x2 2D torus SCI network connections, provided in the High-Performance Computing and Simulation Laboratory. One torus uses the Dolphin

driver with SISC API version 1.3.1; the other uses the Scali 3.1 driver with ScaMPI.

5.1.2 Experiments

This section outlines the experiments performed. Figure 10 shows the actual results each test measured.

5.1.2.1 SCI Raw

SCI DMA raw performance is obtained by executing Dolphin's *dmabench* included in the SISC API installation. For PIO, a program using the SISC API is used to transmit various sizes of data back and forth between two nodes. The latency value is obtained through stream latency for DMA and ping-pong latency for PIO with 1000 repetitions. Latency is calculated as half of the average value obtained. These tests serve as the baseline for comparison.

5.1.2.2 MPI Conduit

The GASNet MPI conduit is installed on the testbed using ScaMPI V3.1 as its MPI layer. The GASNet system provides a test of its AM functions by using them to send a packet of data between two hosts (essentially ping-pong). No data in the message is actually handled except to send the data back to its source. This ping-pong test (1000 repetitions) was used and the final latency values are taken to be half of the average value obtained. This test is used to compare our SCI conduit performance with an existing conduit.

5.1.2.3 SCI Conduit

Latency is measured by clocking each AM send separately. Each send waits for a reply of the same size from the destination node (ping-pong). Final latency values are obtained by taking the average latency value for 5000 sends.

5.2 Results and Analysis

Each of the three types of Active Messages present unique insights to the performance of GASNet and its conduits. Each of the three types appears below in its own subsection.

5.2.1 Short AM Result

Our results show 5 μ s latency for sending a short Active Message on the SCI conduit. Comparing this result to SCI Raw PIO at 80 bytes (which is the size of the header sent) our conduit suffers an overhead of 1-2 μ s. The difference from

the 3.16 μ s obtained from the raw SCI is from packaging the message and determining the correct virtual address to use. An additional timer was placed before and after the actual SCI operation and the results show that creating a suitable command structure before transfer is the main additional cost in our conduit. This overhead cannot be avoided since this process is necessary before any transfer can occur. Nevertheless, when comparing to latency with the MPI conduit, ~15 μ s, our SCI conduit is significantly better and this case is mainly due to the fact that ScaMPI employs an import-as-needed strategy. This strategy incurs a runtime startup overhead for transferring messages of any size.

5.2.2 Medium AM

As with short AM, our SCI conduit outperforms the MPI conduit due to the same reasons mentioned above. The SCI conduit still has the overhead due to packaging of command messages (headers, flags) and determining the correct virtual address, but the overhead does not cause latency to increase as fast as the MPI conduit.

The transmission of the Command Message and Payload of the medium message can be done in two ways. First, the Command and Payload can be copied into one contiguous memory location and then transmitted in one transfer to the receiver. Second, the Command can be transferred to the receiver and then the Payload can be transferred. The second approach introduces a second transmission, but avoids a possibly costly memory copy (context switches to the kernel to perform the copy, then back to the user to perform the transfer).

Originally, medium AM send was implemented in such a way that the payload was copied into memory location right after the header so that the entire message could be transferred with one PIO operation (1 copy, 1 transfer, Figure 12). One would expect this to perform better than two transfers (one for header and another for payload without copying) given that network communication cost is generally much higher than local processing cost. However, our results indicate that using the 0 copy, 2 transfers method is actually more efficient. Part of the reason why the 2-transfer protocol is faster has to do with the memory copy. The SISC API is a user-level API to directly manipulate SCI hardware. Thus, any transactions on the network must originate and terminate in user space. However, memory copies are a kernel space operation by the OS. So in order for a memory copy then a SCI transaction to occur, two context switches are needed. The first switch comes from user space to kernel space to allow the memory copy; the second switch is from kernel space back to user space to initiate the transfer. By remaining only in user space but using two

transfers, the second protocol avoids the costly context switches and OS operations. Additionally, SCI allows up to 16 outstanding transactions to be posted at once. Because of this behavior, we believe that the second SCI transaction overhead is hidden from the user by the first transaction (overlapping transactions).

5.2.3 Long AM

To minimize DMA setup overhead, we use a single, local DMA queue (created in physical memory) that is set up at initialization time. With this approach, the sending of AM long payload can be achieved by copying the source data into the DMA queue and using the DMA queue to perform the SCI network transfer. Our results suffer some performance degradation as compared to SCI Raw. We are currently working on optimizations to minimize the overhead.

6 Conclusions and Future Work

GASNet is an important part of the push to expand UPC programming capabilities to clusters. The GASNet conduits available on many networks allow UPC to be executed on a wide variety of platforms. SCI is a high-performance network that has many features that can be used to efficiently execute GASNet and UPC. By extending GASNet to SCI through the creation of an SCI conduit, the availability of UPC to parallel programmers increases. The creation of the GASNet Core API is an essential first step in accomplishing this goal.

Our preliminary results show that the SCI conduit is a promising extension to the GASNet system, and that it can outperform the MPI conduit using Scali API. Several possible enhancements are under investigation to improve the performance of the SCI conduit. Care is needed in balancing the many different aspects of network performance so that the SCI conduit can fully exploit the features available in the SCI network.

Currently, the SCI conduit only supports a GASNet global segment size up to 1MB (equal to the size of the Long AM payload segment). This restriction is primarily due to the limitation of the current SISCO API where the size of each local/remote segment cannot exceed 1MB (under Linux, without applying a large physical area patch). We are currently working with Dolphin Inc. with the hope of resolving this issue.

After the complete GASNet SCI Conduit Core API is tested, the next step is the design and implementation of the GASNet SCI Conduit Extended API followed by user-level performance benchmarking. The conduit implementing the Extended API should be able to bypass all unnecessary Core API calls and access the SISCO API directly to achieve better performance.

7 Acknowledgements

We thank Dan Bonachea and the UPC group at UC Berkeley for their support and the opportunity to work with them on this project. Also, we thank Hugo Kohmann at Dolphin Interconnect Solutions, Inc. for technical support with the SCI equipment and drivers.

8 Figures

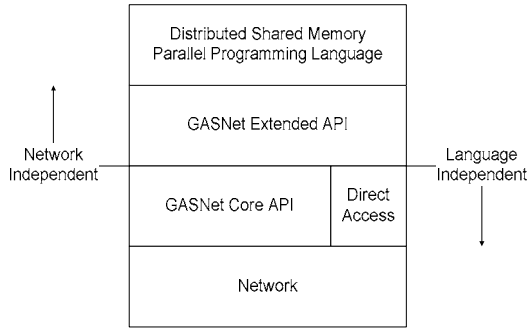
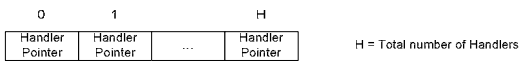
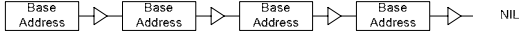


Figure 1 – GASNet Layers

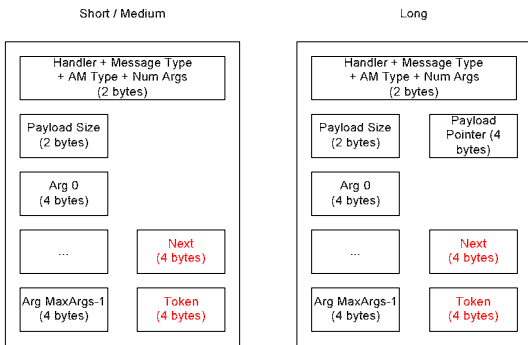
A. Handler Table



B. LSP/RSP



C. Command Structures



D. Work Queue



Figure 2 – SCI Conduit Core API Data Structures

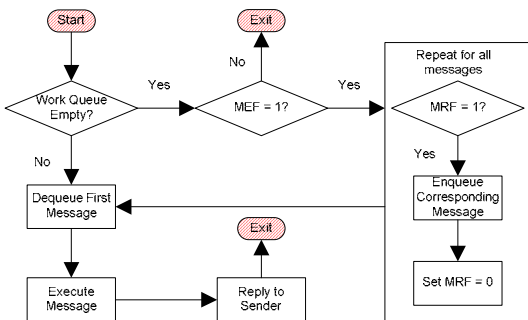


Figure 3 – Message Polling Flow Chart

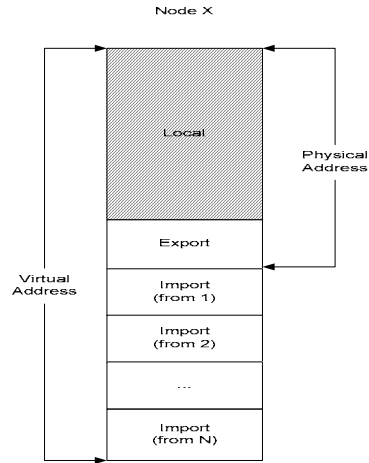


Figure 4 – Memory Mapping

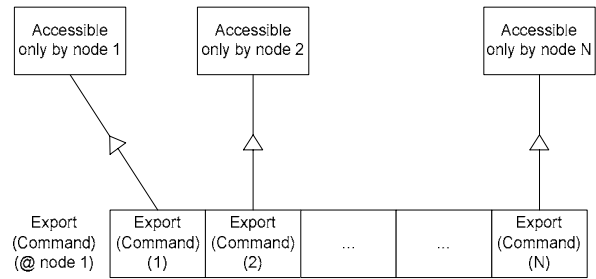


Figure 5 – GASNet Command Segments in an SCI node

N = Number of Nodes in System

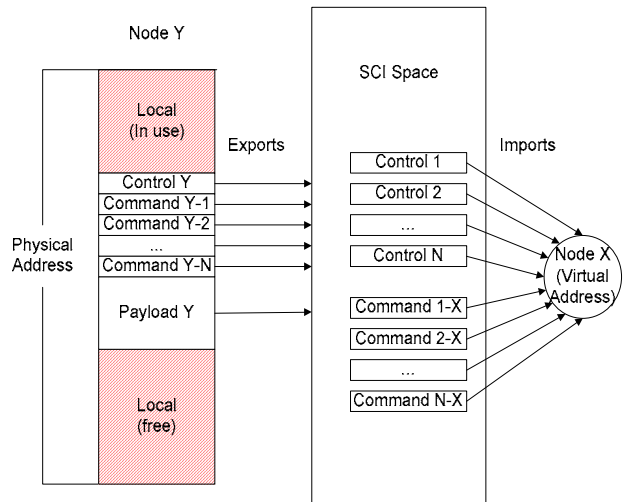


Figure 6 – Segment Exportation and Importation during Initialization

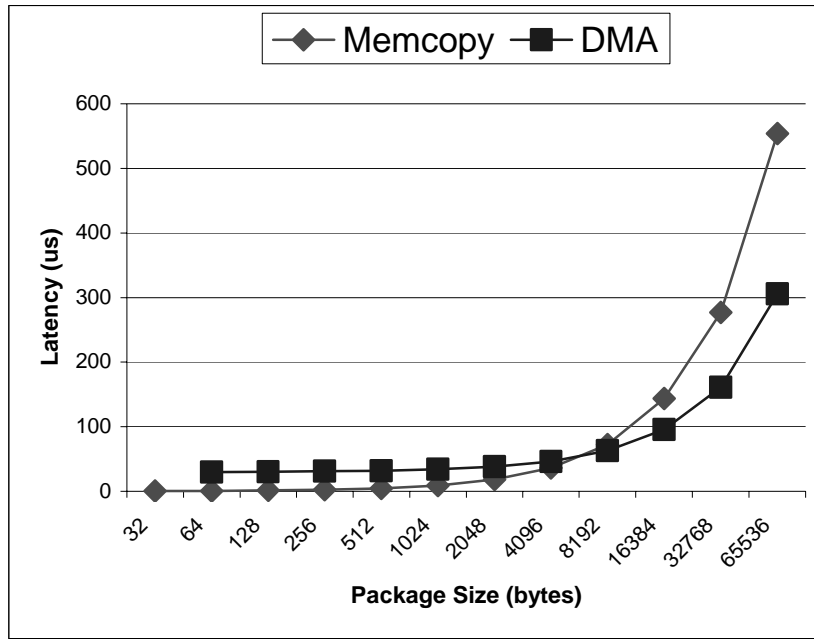


Figure 7 – Performance Results for SCI Shared-Memory and DMA Operations

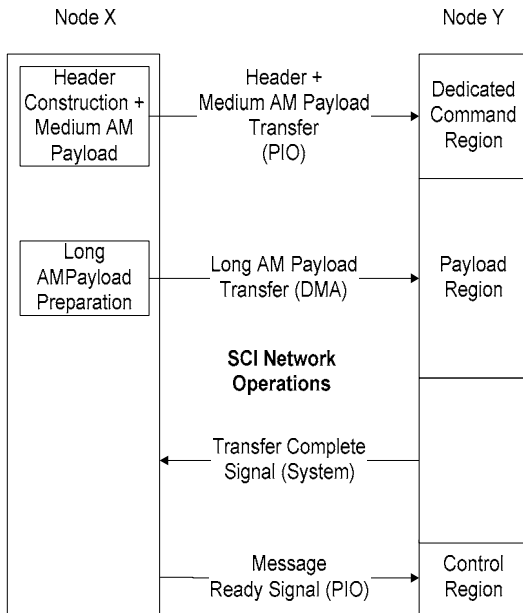


Figure 8 – Sending Message from Node X to Node Y

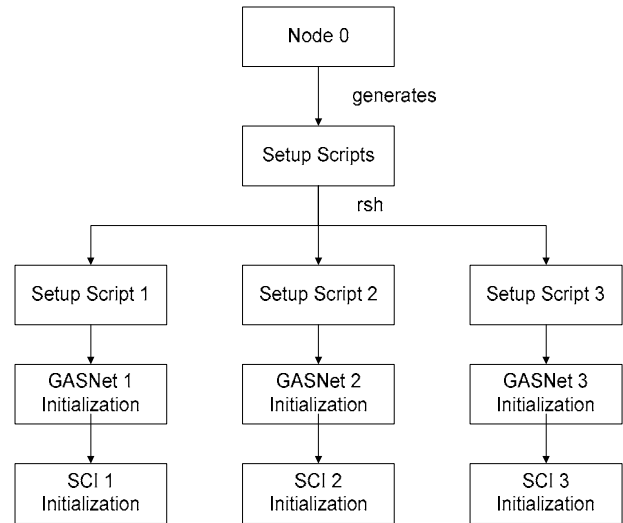


Figure 9 - Bootstrapping Using rsh

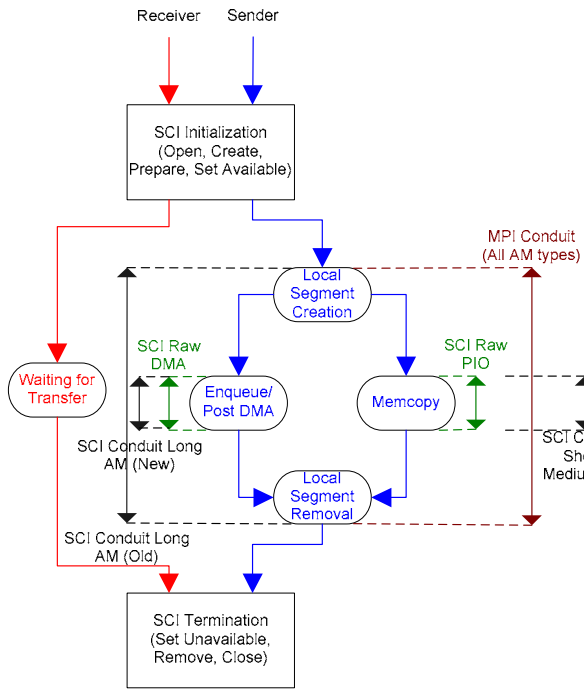
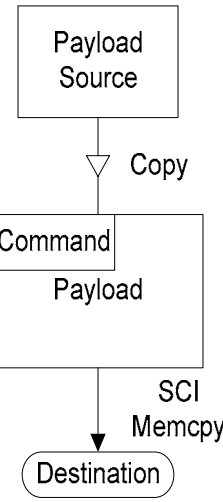


Figure 10 – Experiment Software Flow with Indication of Measurement Attributes

1 Copy, 1 Transfer



0 Copy, 2 Transfers

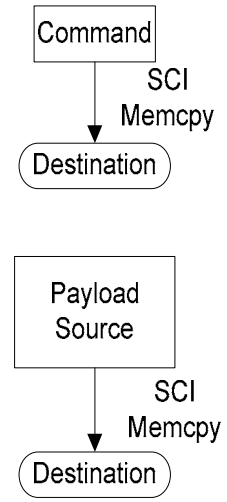


Figure 12 – Possible Transfer Mechanisms for Medium AM

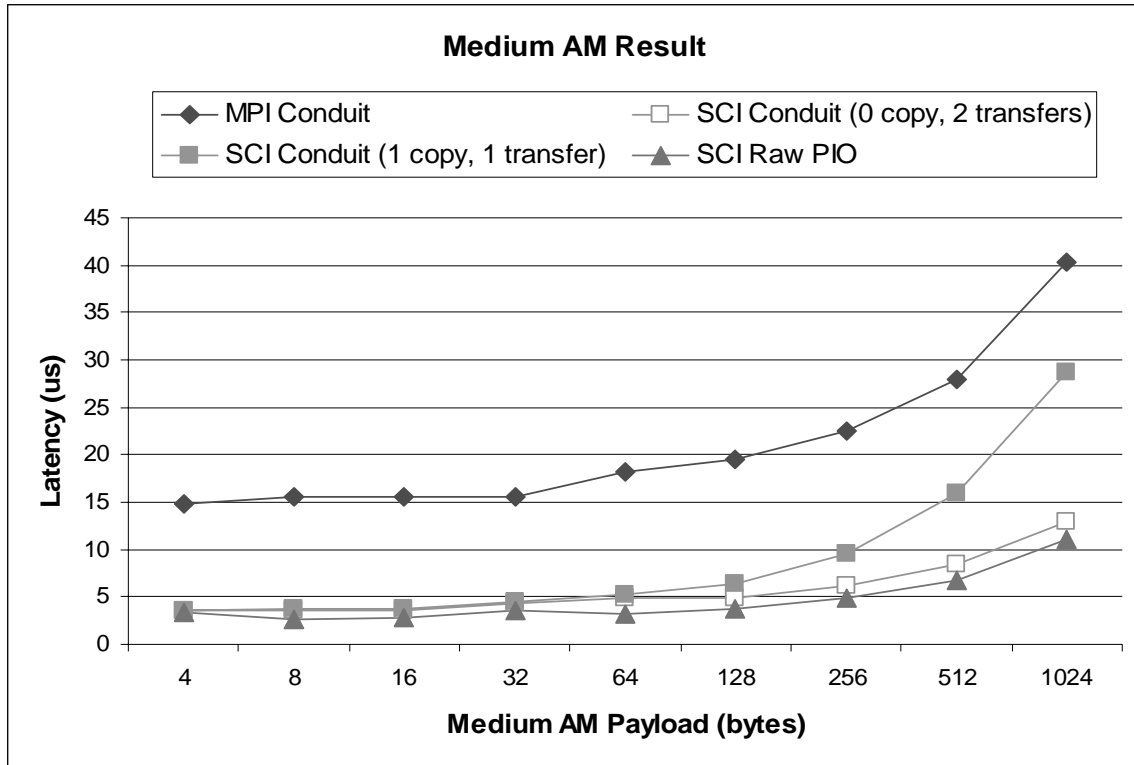


Figure 11 – Medium AM Results

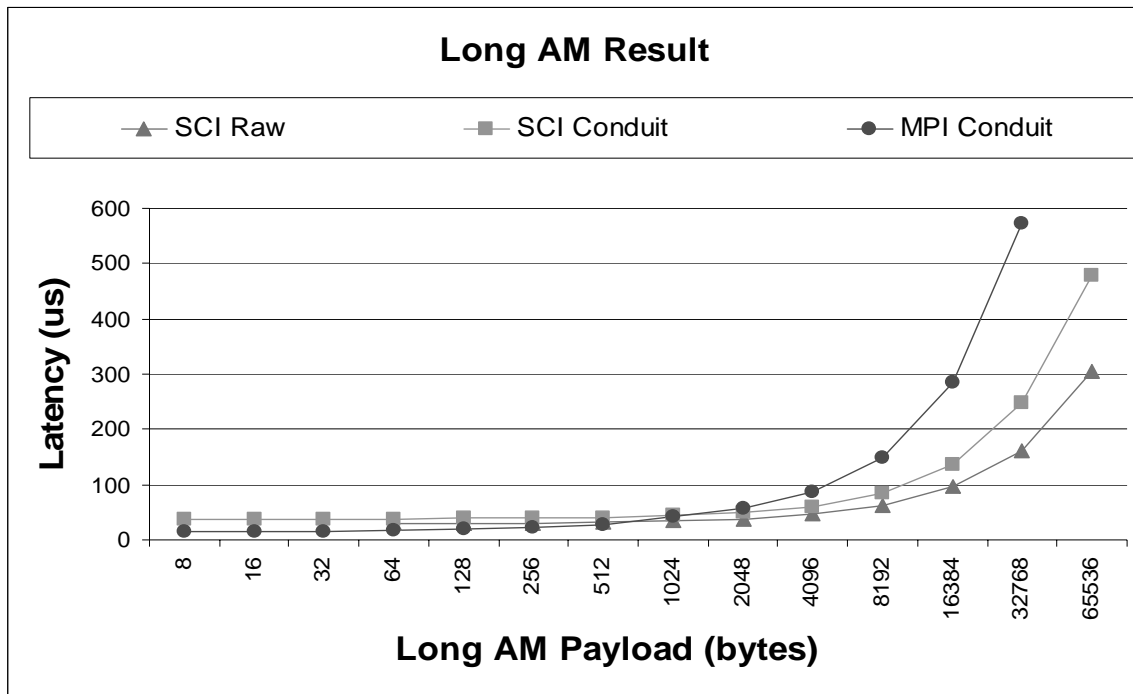


Figure 13 – Long AM Results

9 References

1. W. Carlson, J. Draper, D. Culler, K. Yelik, E. Brooks, K. Warren, "Introduction to UPC and Language Specification", May 1999 <http://www.gwu.edu/~upc/pubs.html>.
2. D. Gustavson and Q. Li, "The Scalable Coherent Interface (SCI)", *IEEE Communications*, Vol. 34, No. 8, August 1996, pp. 52-63.
3. D. Gonzalez, A. George, and M. Chidester, "Performance Modeling and Evaluation of Topologies for Low-Latency SCI Systems", *Microprocessors and Microsystems*, Vol. 25, No. 7, Oct. 2001, pp. 343-356.
4. Dolphin Inc., "SISCI API User Guide", May 2001, <http://www.dolphinics.com/support/documentation.html>.
5. D. Bonachea, "GASNet Specification Version 1.3", April 2003 <http://www.cs.berkeley.edu/~bonachea/gasnet/dist/docs/gasnet.pdf>.
6. C. Bell and D. Bonachea, "A New DMA Registration Strategy for Pinning-Based High Performance Networks", Workshop on Communication Architecture for Clusters (CAC'03), 2003 <http://www.cs.berkeley.edu/~bonachea/upc/bell-bonachea-firehose.pdf>.
7. Unified Parallel C <http://www.upc.gwu.edu/>.
8. A. Mainwaring and E. Culler, "Active Messages: Organization and Applications Programming Interface", Technical Document, 1995 <http://now.cs.berkeley.edu/Papers/Papers/am-spec.ps>.
9. M. Schulz, J. Tao, C. Trinitis, and W. Karl, "SMiLE: An Integrated, Multi-Paradigm Software Infrastructure for SCI--based Clusters", 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), Berlin, Germany, May, 2002.
10. M. Ibel, K.E. Schauser, C. J. Scheiman, and M. Weis, "Implementing Active Messages and Split-C for SCI Clusters and Some Architectural Implications", Sixth International Workshop on SCI-based Low-cost/High-performance Computing (SCIzzL-6), Santa Clara, CA, September 1996.
11. M. Ibel, M. Schmitt, K.E. Schauser, A. Acharya, "Shared Memory vs. Message Passing on SCI: A case study using Split-C", SCI: Scalable Coherent Interface, Springer Lecture Notes in Computer Science 1734, 1998.
12. Official GASNet website <http://www.cs.berkeley.edu/~bonachea/gasnet>.