

# SCI Networking for Shared-Memory Computing in UPC: Blueprints of the GASNet SCI Conduit

H. Su, B. Gordon, S. Oral, A. George

*High-performance Computing and Simulation (HCS) Research Lab, Dept. of Electrical and  
Computer Engineering, University of Florida, Gainesville, Florida 32611-6200  
{su, gordon, oral, george}@hcs.ufl.edu*

## Abstract

*Unified Parallel C (UPC) is a programming model for shared-memory parallel computing on shared- and distributed-memory systems. The Berkeley UPC software, which operates on top of their Global Addressing Space Networking (GASNet) communication system, is a portable, high-performance implementation of UPC for large-scale clusters. The Scalable Coherent Interface (SCI), a torus-based system-area network (SAN), is known for its ability to provide very low latency transfers as well as its direct support for both shared-memory and message-passing communications. High-speed clusters constructed around SCI promise to be a potent platform for large-scale UPC applications. This paper introduces the design of the Core API for the new SCI conduit for GASNet and UPC, which is based on Active Messages (AM). Latency and bandwidth data were collected and are compared with raw SCI results and with other existing GASNet conduits. The outcome shows that the new GASNet SCI conduit is able to provide promising performance in support of UPC applications.*

Keywords - Scalable Coherent Interface, Global Address Space Networking, Unified Parallel C, Active Messages.

## 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 advent of parallel computing. To supplement the architectural improvements in this area, parallel programming models have emerged to provide programmers alternate ways

in solving complex and computationally intensive problems. Such models include message passing, shared memory, and global address space.

While message passing and shared memory are the two most popular ways to implement parallel programs, global address space is quickly gaining momentum. One of the reasons for this development is the growing acceptance of Unified Parallel C (UPC) [1,2] and other models like it. UPC is a parallel extension to the ISO C standard that gives programmers the ability to create parallel programs that can target a variety of parallel architecture platforms while maintaining a familiar C-style structure. This approach allows a smaller learning curve for people with C experience to begin creating parallel programs and often results in tighter and more efficient code.

One recent development in UPC is the interest in providing a means for executing UPC over commercial-off-the-shelf (COTS) clusters. The Berkeley UPC runtime system [3], developed by U.C. Berkeley and Lawrence Berkeley National Laboratory (LBNL), is a promising tool now available to support this endeavor. An underlying key to this system is the Global Addressing Space Networking (GASNet) communication system [4,5]. GASNet defines a standard application interface that can be implemented over a wide variety of standard and high-performance networks such as Ethernet, InfiniBand, Myrinet, and Quadrics.

In this study, we present the design of a new GASNet conduit operating over the Scalable Coherent Interface (SCI) network [6]. Benchmarks were executed on the newly developed conduit and compared against the raw performance of SCI, the GASNet Myrinet conduit, and GASNet MPI conduit on SCI using Scali's ScaMPI [7] to evaluate various strengths and weaknesses.

The next section of this paper briefly describes the architecture of SCI and GASNet. In Section 3, we discuss related research. Section 4 describes the design

overview of the GASNet/SCI conduit. Section 5 presents the performance results and analyses. Finally, Section 6 presents conclusions and directions for future research.

## 2. Background

In the following subsections we present an overview of the SCI high-performance network. Also included is a brief introduction to the GASNet communication system.

### 2.1. SCI

SCI is an ANSI/ISO/IEEE standard (1596-1992) that describes a packet-based protocol [8] for system-area networking. SCI was initially developed as an attempt to address the problems associated with buses for use with many processors. It has evolved to become a high-performance interconnect for SANs and embedded systems. SCI uses point-to-point links, maintaining low latency while achieving high data rates between nodes. It features a shared-memory mentality so that memory on each node can be addressable by every other node on the network. SCI uses 64 bits in its addressing. The most-significant 16 bits are used to specify the node in the network, and the remaining 48 bits are used for addresses within each node. With this scheme, the SCI environment can support up to 64K nodes with 256TB of addressable space.

SCI offers many advantages for the unique nature of parallel computing demands. Perhaps the most significant of these advantages is its low-latency performance, typically (based on current commercial products from Dolphin) on the order of single-digit microseconds for remote-write operations and tens of microseconds for remote-read operations. Based on the latest technology, SCI offers a link data rate of 5.3 Gb/s with topologies including 1D (ring), 2D, or 3D torus.

The Dolphin SISI API [9] is a standard set of API calls allowing users to access and control SCI hardware behavior directly based on a shared-memory paradigm. To enable inter-node communication, the receiver must set aside a portion of its physical memory (global memory region) for use by the SCI network. The sender then imports the memory region into its virtual address space and is thus able to read and write the receiver's memory region by way of either PIO (shared-memory operation) or DMA (zero-copy operation) transfer modes. The SCI hardware automatically converts accesses in SCI-mapped virtual address space to network transfers.

### 2.2. GASNet

Global Addressing Space Networking (GASNet), developed at UCB/LBNL, is a language-independent, low-level communications layer that provides network-independent, high-performance communication primitives aimed at supporting parallel shared-memory programming languages such as UPC and Titanium, a parallel dialect of Java. The system is divided into two layers, the GASNet Core API and the GASNet Extended API (Figure 1). The Core API is a narrow interface based on Active Messages (AM) [10] and the network-specific Firehose memory registration algorithm [11]. The Extended API is a network-independent interface that provides medium- and high-level operations on remote memory and collective operations.

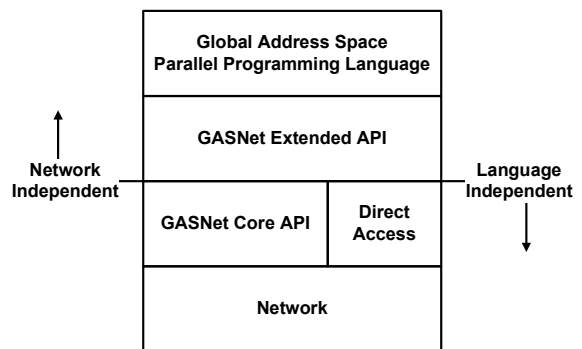


Figure 1. GASNet layers overview

The GASNet segment is the location where most of the GASNet operations target. There are three ways the segment can be configured, as *fast*, *large*, or *everything*. Under the *fast* configuration, the size of the segment may be limited to provide faster transfers of GASNet operations. The *large* configuration makes a large portion of memory available to the segment. The size may include all of the physical memory or more. The *everything* configuration makes the whole virtual address space on every node available for GASNet operations.

Currently, GASNet supports execution on UDP, MPI, Myrinet, Quadrics, InfiniBand and IBM LAPI. GASNet was first released on 1/29/2003 with the latest release as of this writing being Version 1.3.

## 3. Related research

Since our GASNet Core API must provide for AM over SCI, Ibel's paper [12] is useful as it discusses several possible ways to execute AM over SCI. However, his simple remote-queue implementation poses several limitations. First, with 1 buffer space

for all AM replies, each node is restricted to having only 1 outstanding AM request to the whole network at any given time. Furthermore, the need for the receiver to copy bulk data (long AM payload) from the 4KB buffer to its appropriate memory location, and the cost of message polling ( $O(N)$ , where  $N$  denotes the system size), introduce additional overhead that significantly impacts system performance. With applications that exhibit frequent inter-node communication, system performance is degraded to a degree that the benefit of parallelization is no longer observed.

Ibel briefly described a split remote-queue scheme that uses circular queues of  $Nk$  (where  $k$  is a constant) messages (one queue for each node able to hold  $k$  messages) to allow parallel sending and receiving of messages. Unfortunately, this approach is not deadlock-free and the overhead for copying bulk data and message polling still remains.

Additional research that was instrumental to this work consists of other existing GASNet conduits. The design documents and source code available on the GASNet website [4] were used as a guide in the design of the Core API for the new SCI conduit.

## 4. Core API design

SCI hardware is designed such that remote writes are  $\sim 10$  times faster than remote reads. This disparity is due to the inability to streamline reads through the memory PCI bridges. As a result, to obtain the best performance, only remote writes are used in our conduit design, as in Ibel's approach. Additionally, due to driver limitations, only the GASNet *fast* segment configuration is supported by the SCI conduit. Future improvements will allow support of the other configurations.

### 4.1. Basic communication regions

Instead of only 1 buffer space for both AM requests and replies as in Ibel's split-queue scheme, we divide the buffer (command region) into a request and a reply queue of equal size making the system deadlock free. Each request/reply buffer space is set to be the size of the longest AM header plus the maximum size of a medium payload. The request and reply are paired so that a node with an outstanding request to another node is guaranteed to have space to hold the reply for that particular request. Each node has a message queue reserved for it on every other node. This scheme allows each node to locally manage outgoing messages and guarantee no conflicts with other nodes (Figure 2).

Similar to Ibel's approach, a message-ready flag is used to indicate if a particular message exists in a queue or not. However, rather than attaching the flag to the end of the AM message, these flags are separately placed in an array (control region) that is accessible by all other nodes. This method provides better data locality when checking for new messages, as all the message-ready flags now reside in one contiguous memory region. In addition, a single global message-exist flag is used to indicate the existence of any new messages.

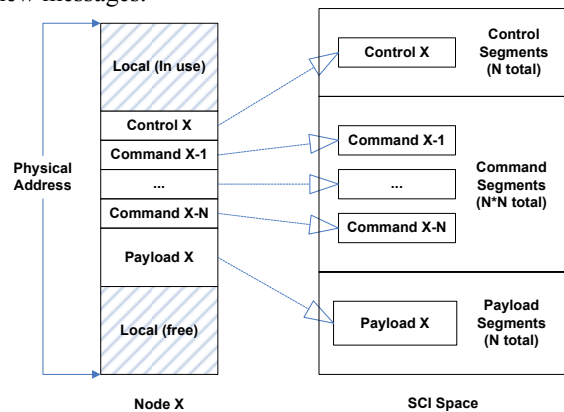


Figure 2a. Conceptual diagram of the segment exportation mechanism

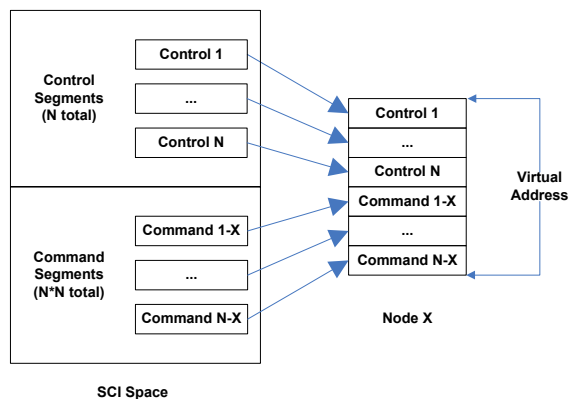


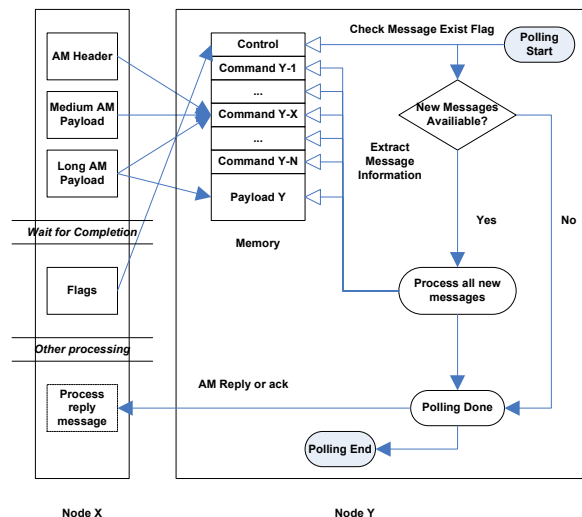
Figure 2b. Conceptual diagram of the segment importation mechanism

Finally, the size of the long AM payload region is significantly bigger and it corresponds to the range of remotely accessible memory as specified by the GASNet *fast* segment configuration which the user defines, thus minimizing unnecessary data copying. Since the importing of regions occupies local virtual address space equal to the size of the segment, the large payload segments (payload regions) are not imported at initialization time so as to improve scalability. Fortunately, DMA transfer mode allows communication to

take place without having to import the region into virtual memory space, but with added overhead.

## 4.2. AM communication

The message sending and handling process is illustrated in Figure 3. In order to send a message from a sender node to a receiver node, the sender first prepares the AM header, which contains information such as the handler to be called, message type, payload size, etc. Once prepared, the header is then written to the receiver's command region using a PIO transfer. For a medium AM message, another remote PIO write operation is used to transfer the medium payload to the same command region. The same sequence of operations is used for long AM transfers to handle the unaligned portion of the long payload (see Section 5.2.2 for further explanation). Otherwise, the data is sent directly to the payload region via a DMA transfer.



**Figure 3. High-level flowchart for inter-node communication**

Upon completion of these transfers, the sender writes the two message flags to the receiver's control segment. The message-exist flag is used to tell the receiver that there is at least one new message available and the message-ready flag indicates that a particular message buffer contains a message. When the receiver calls the polling process, it checks the message-exist flag to see if there are any new messages that need to be handled. If there are, the receiver scans message-ready flags and handles the appropriate newly arrived messages. Using this approach, the cost of an unsuccessful poll is  $O(1)$  and  $O(N)$  for a successful poll, leading to amortized costs for polling of only  $O(1)$ .

## 5. Results and analysis

In this section we present the latency and bandwidth results of the first full design and implementation of our Core API. These results are compared against Dolphin SISCi raw performance and two other existing GASNet conduits, namely the GM conduit for Myrinet and the MPI conduit, a core-only implementation on SCI using Scali's ScaMPI. ScaMPI is a commercial MPI implementation for SCI, and it is considered the most efficient communication layer implemented to date for SCI. This comparison of results is used to evaluate the performance of our design.

The GASNet system provides a reference-extended API implementation that is based on Core API functions. Consequently, a complete and fully functional GASNet conduit is created with the successful completion of the Core API. To complete the analysis of our design, we compared the results of the basic Extended API operations *put* and *get* for our native SCI conduit against the MPI conduit executing on top of ScaMPI.

### 5.1. Experimental setup

Here we describe the environment and testing procedures used in obtaining performance measurements from each of the software environments.

**5.1.1. Testbed.** Two sets of machines were used in this study. The first set consists of 16 server nodes, each with dual 2.4GHz Intel P4 Xeon CPUs with 256KB L2 cache, 1GB of DDR PC2100 (DDR266) RAM, and a 533MHz system bus. Each node is equipped with a Dolphin D339 3D SCI card and uses Linux Red Hat 9.0 with kernel 2.4.20-8smp and gcc version 3.3.2. These SCI nodes are wired and configured as two  $4 \times 2$  2D torus networks. One torus uses the free open-source driver with SISCi API V2.2 provided by Dolphin, and the other uses the commercial Scali V4.0 driver with ScaMPI.

Michigan Technological University (MTU) graciously provided access to their Myrinet 2000E cluster for this work. Their cluster consists of 16 server nodes, each with dual 2.2GHz Intel P4 Xeon CPUs with 256KB L2 cache, 2GB of DDR PC2100 (DDR266) RAM, and a 533MHz system bus. A 16-port Myrinet 2000 switch is used to connect these nodes. The Myrinet NIC in each node features an onboard 133MHz LANai 9.0 CPU with 2MB of on-card memory using GM V1.6.3.

**5.1.2. Experiments.** Performance results for SCI Raw are obtained using *scipp* (*PIO benchmark*, *ping-pong*)

and *dma\_bench* (*DMA benchmark, one-way*), latency and bandwidth benchmarks provided by Dolphin for the SISC API. Conduit results are obtained by executing a slightly modified version of *testam* benchmark from the GASNet test suite. The *testam* code was changed only to output the bandwidth measurements for AM long transfers.

To test the latency of small-message *put/get* operations in GASNet, we use the *testsmall* benchmark from the GASNet test suite. It uses the *gasnet\_put()* and *gasnet\_get()* functions to send data back and forth between nodes, obtaining the round-trip latency for these requests. Bandwidth is measured using the *testlarge* benchmark available in the GASNet test suite. It uses the various bulk-data transfer functions available in the Extended API to send one-way data between two nodes.

## 5.2. Core API AM results and analysis

Short, medium, and long AM latency, as well as long AM bandwidth results, are shown in this section. As short and medium AM transfers are typically small in size and do not transfer large amounts of data, bandwidth numbers for them are not included. Comparison and analysis of our SCI conduit's performance versus the SCI Raw, the MPI/ScaMPI Conduit, and the Myrinet Conduit are also discussed. Unfortunately, direct comparisons between our results and those from Ibel's work cannot be made due to vastly different hardware/software testbeds.

**5.2.1. Short/Medium AM.** Compared to SCI raw performance, our SCI conduit adds ~12us of overhead (Figure 4). The main cause is the overhead added to package and unpackage the AM header, obtaining free buffer space and system sanity checks. Our results are comparable to the Myrinet conduit, but somewhat lags behind the MPI/ScaMPI conduit. Other possible causes for the overhead and reason why MPI/ScaMPI has better performance is still under investigation.

The transmission of medium AM messages can be performed in two ways. The header and payload can be copied into one contiguous memory location and then transmitted in one transfer to the receiver, or instead the header and payload can be transferred separately to the receiver (Figure 5). One would expect the first approach to perform better than the second given that network communication cost is generally much higher than local processing cost. However, our testing indicates that the performance of the "2 network transactions" mechanism is comparable and in most cases (< 64B and > 1024B) slightly more efficient than the "1 network transaction" mechanism (Figure 6). One reason for this may have to do with the removal of

*memcpy()*, which can sometimes be an expensive operation. Another part of the reason may be that SCI allows up to 16 outstanding transactions to be posted at once. Because of this, the second SCI transaction overhead is partially hidden from the user by the first transaction (i.e. overlapping transactions).

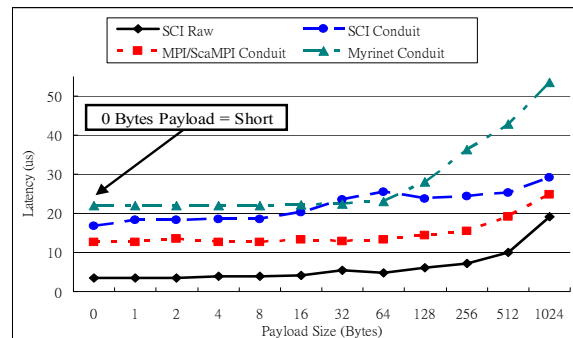


Figure 4. Short/Medium AM ping-pong latency results

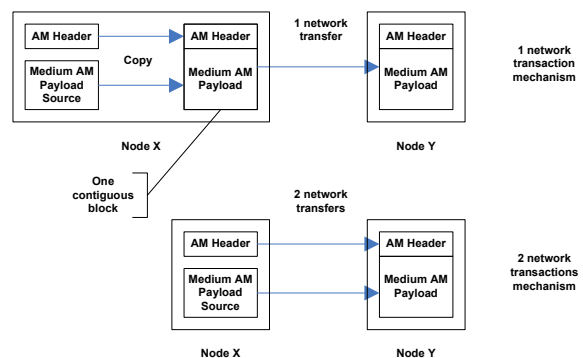


Figure 5. Conceptual diagram of "1 network transaction" and "2 network transactions" message delivery mechanisms

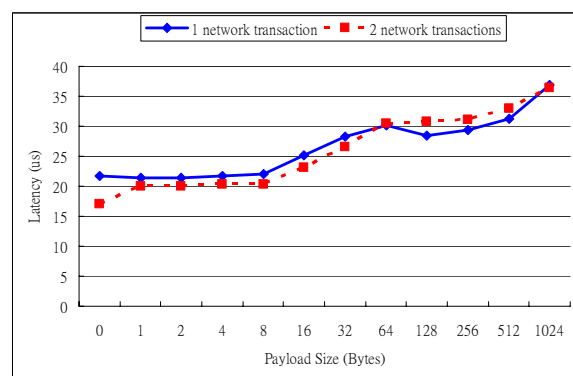


Figure 6. Performance comparison of "1 network transaction" and "2 network transactions" message delivery mechanisms

**5.2.2. Long AM.** The SISC API requires any DMA transfer to have 8-byte alignment between the source and the target segment (both starting address and transfer size). Sending of unaligned data thus became a problem as costly dynamic mapping (~200us overhead) and unmapping of the target segment is needed. To overcome this shortcoming, the request/reply buffer region reserved for medium payload is used as a bounce buffer for the unaligned portion of the long payload, which is later copied to the appropriate payload address when handled by the receiver. Furthermore, because of the high DMA engine setup overhead (~30μs), any long payloads that are less than 2048 bytes are treated as unaligned data and written to the command segment using PIO mode instead. In doing so, our conduit is able to achieve better performance for small long AM payloads and suffer lower overhead for unaligned data transfers (~13us). Future implementations of the SCI conduit might switch back to use the DMA engine directly, since Dolphin is currently working on improving their driver to reduce the mapping overhead, DMA engine start-up overhead, and the alignment requirement.

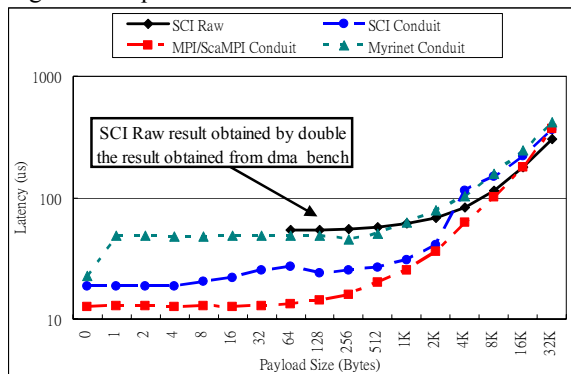


Figure 7. Long AM ping-pong latency results

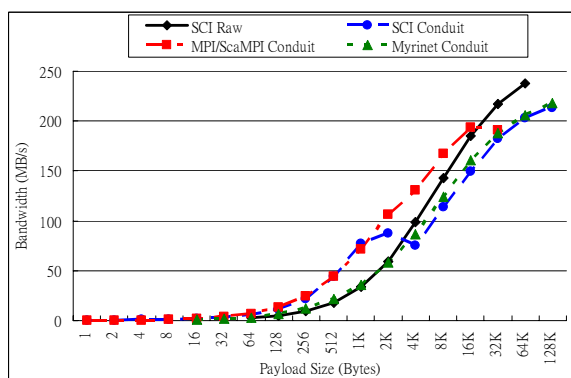


Figure 8. Long AM bandwidth results

Our long AM latency (Figure 7) and bandwidth results (Figure 8) follow the same growth trend as that of SCI Raw and are comparable to the Myrinet conduit.

Although MPI/ScaMPI has better performance for smaller payload size, its maximum bandwidth is about 190 MB/s, mainly due to the fact that it uses PIO exclusively, whereas our conduit rises to 213 MB/s with payload size of 128K.

### 5.3. Put/Get

There are two modes of *testsmall*, transfers to within and without the main GASNet segment. Since all small and medium AM transactions take place through buffers, the results for both modes are the same and only the graph for transfers within the segment is shown. Figure 9 shows the results of *testsmall* for our SCI conduit and the MPI conduit on ScaMPI. Since the Extended API implementation of these two conduits is based on AM transactions in their Core APIs, the results correspond almost exactly to the latency gathered for the small and medium AM transfers in the Core API. The spike observed for MPI/ScaMPI (in) is not a measurement anomaly as multiple runs yields the same pattern. The reason behind this anomaly is currently under investigation.

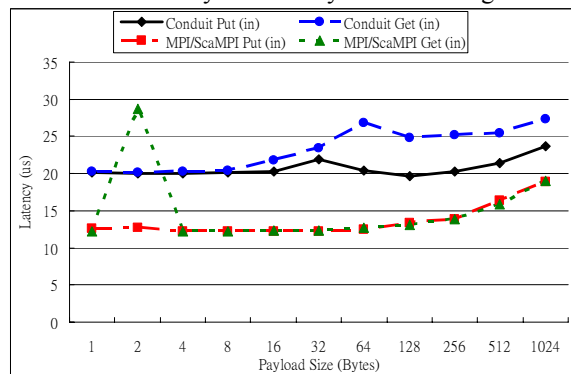


Figure 9. Put/Get latency results

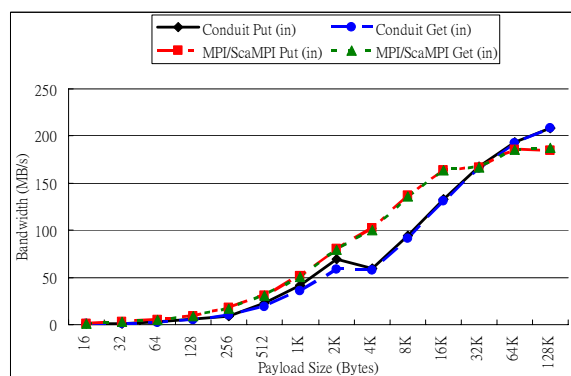


Figure 10. Put/Get bandwidth results

The results for all blocking and non-blocking functions were the same, so only the results for *gasnet\_put\_bulk()* and *gasnet\_get\_bulk()* are shown here.

Similar to *testsmall*, there are two modes of transfer in *testlarge*. Because our Core API currently supports only the *fast* segment configuration, it is optimized for transfer of data that falls within the main GASNet segment. Therefore, only the results for one-way, in-segment transfers are shown in Figure 10.

Similar to large AM transfers, the MPI conduit using ScaMPI achieves slightly better bandwidth for smaller transfer sizes. However, for transfers of 32KB and more, our SCI conduit shows better performance.

## 6. Conclusions

GASNet is an important part of the push to expand UPC shared-memory computing capabilities to network-based systems like 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 step in accomplishing this goal, as a complete Core API implementation is sufficient for a GASNet conduit.

The tests conducted show that we have designed and created a complete and potent GASNet conduit design for SCI. The performance of our SCI conduit is shown to be comparable to the Myrinet conduit and slightly behind the MPI/ScaMPI conduit which uses proprietary SCI driver and MPI software. This outcome strengthens our belief that our SCI conduit is a promising extension to the GASNet system, as the driver used in the creation of the SCI conduit is free and open-source.

Several ideas are under investigation which will further improve the performance of our conduit. Care is needed in balancing the many different aspects of network performance so that the SCI conduit can fully exploit the unique features available in the SCI network. Furthermore, currently the SCI conduit only supports GASNet global segment sizes up to 2MB, under Linux, without applying a large physical area patch. This requirement limits the usage of our conduit to those clusters whose system administrators are willing to patch the kernel on each SCI node. This patch requirement is primarily due to the limitation of the current SISCO driver where the size of each segment needs to be physically contiguous and relies on the underlying operating system to ensure continuity. We are currently working with Dolphin to resolve this issue and increase the ease of use of this conduit.

Initial testing at the GASNet put/get level with our Core API again indicates that our conduit is comparable to other conduits. We are currently completing the design and implementation of an Extended API in order to improve the performance of our SCI conduit. Once complete, benchmarks at the UPC application level will be used to obtain a better assessment of the effectiveness of our SCI conduit from the communication to the application layer.

## 7. Acknowledgements

This work was supported in part by the U.S. Department of Defense and by equipment support of Dolphin Interconnect Solutions Inc. Also, we would like to express our thanks for the helpful suggestions and cooperation of Dan Bonachea and the UPC group members at UCB and LBNL, Hugo Kohmann and the support team at Dolphin for technical assistance, and to Steven Seidel's group at MTU for providing access to their Myrinet 2000E cluster

## 8. 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] Official Unified Parallel C website. <http://www.upc.gwu.edu/>
- [3] Official Berkeley UPC website. <http://upc.nersc.gov/>
- [4] Official GASNet website. <http://www.cs.berkeley.edu/~bonachea/gasnet>
- [5] D. Bonachea, "GASNet Specification Version 1.3", April 2003. <http://www.cs.berkeley.edu/~bonachea/gasnet/dist/docs/gasnet.pdf>
- [6] D. Gustavson and Q. Li, "The Scalable Coherent Interface (SCI)", *IEEE Communications*, Vol. 34, No. 8, August 1996, pp. 52-63.
- [7] Scali, "ScaMPI – Design and Implementation". <http://www.scali.com/whitepaper/other/scampidesign.pdf>
- [8] IEEE Service Center, "Scalable Coherent Interface, ANSI/IEEE Standard 1596-1992", Piscataway, New Jersey, 1993.
- [9] Dolphin Inc., "SISCO API User Guide", May 2001. <http://www.dolphinics.com/support/documentation.html>

[10] A. Mainwaring and E. Culler, “Active Messages: Organization and Applications Programming Interface”, Technical Document, 1995.

[11] 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.

[12] 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.