Ultra-High Performance Communication with MPI and the Sun Fire™ Link Interconnect

Steven J. Sistare
steve.sistare@sun.com

Christopher J. Jackson
cjj@sun.com

Sun Microsystems, Inc.

Abstract

We present a new low-latency system area network that provides the ultra-high bandwidth needed to fuse a collection of large SMP servers into a capability cluster. The network adapter exports a remote shared memory (RSM) model that supports low latency kernel bypass messaging. The Sun™ MPI library uses the RSM interface to implement a highly efficient memory-to-memory messaging protocol in which the library directly manages buffers and data structures in remote memory. This allows flexible allocation of buffer space to active connections, while avoiding resource contention that could otherwise increase latencies. We discuss the characteristics of the interconnect, describe the MPI protocols, and measure the performance of a number of MPI benchmarks. Our results include MPI inter-node bandwidths of almost 3 Gigabytes per second and MPI ping-pong latencies as low as 3.7 microseconds.

Keywords: interconnects, MPI, kernel bypass, remote shared memory, SAN, performance evaluation

1. Introduction

The quest to build increasingly powerful cluster configurations for parallel computing has been enabled by the availability of increasingly capable network interconnects and software. Interconnect technologies such as Myrinet provide moderately high bandwidth and support kernel bypass messaging for low latency [1][2]. These interconnects have been primarily utilized in clusters whose compute nodes contain only a few processors each [3][4]. The Berkeley CLUMPS project was one of the first to use larger and more capable SMP systems as the basic building blocks for a cluster [5][6]. SMP systems with larger processor counts make much higher demands of the network interconnect. Indeed, no existing interconnect can supply the bandwidth needed to build a balanced SMP cluster with comparable on-node and off-node bandwidths.

To solve this problem, Sun Microsystems has developed the high performance Sun Fire™ Link interconnect, which allows multiple network interfaces to be striped to provide multi-gigabyte per second transfer rates. It also supports kernel bypass messaging via remote shared memory (RSM), whereby memory regions on one machine can be mapped into the address space of another. We have exposed these capabilities to applications via the Sun MPI library, which implements a number of unique features that

0-7695-1524-X/02 $17.00 (c) 2002 IEEE

take maximum advantage of the characteristics of the interconnect. These features include memory-to-memory messaging, deferred connection establishment, interface striping, and multiple transfer protocols.

In this paper, we discuss the characteristics of the interconnect, present the Sun MPI implementation over remote shared memory, and close with a section on performance results as measured on Sun Fire servers clustered with the Sun Fire Link interconnect.

2. Interconnect Description

This section gives a brief description of the Sun Fire Link cluster interconnect, as well as a comparison to similar interconnect technologies.

2.1 Optical Links

Cluster interconnect traffic is carried over fiber-optic communication links. Each link supports full-duplex, point-to-point communication.

A link comprises a transmitter and receiver at each endpoint, and two unidirectional fiber-optic ribbon cables, which may be up to 20 meters long. Each cable contains twelve multimode optical fibers, operating at 1.2 Gb/s. One fiber is used for a 300 MHz clock, and the rest carry data; four data bits are sent on each fiber per clock cycle.

The transmitter employs a Vertical Cavity Surface Emitting Laser (VCSEL) array operating at 850 nm. Emission levels are kept below critical levels so the links are eye safe (i.e., Class IIIA compliant). The optical receiver employs a high speed InGaAs PIN photodiode array, which is AC coupled to the detection circuitry.

The raw data rate of the link is 1.65 GB/s. After coding, framing and protocol overhead, one link can theoretically sustain 1 GB/s of unidirectional user data bandwidth. For bidirectional traffic, the capacity is 800 MB/s in each direction.

2.2 NI ASICs and Switch

Hosts using the network interconnect contain at least one Sun Fire Link Network Interface ASIC. The NI supports two optical links, and connects directly to the Sun Fireplane system interconnect via a 2.4 GB/s data path. The NI provides a number of instrumentation registers which may be used to analyze remote traffic and performance. These are described more fully in [7].

The Sun Fire Link Switch ASIC is a derivative of the NI. Its two internal crossbars switch traffic at full link bandwidth amongst up to eight optical links.

The Sun Fire Link Switch allows up to eight hosts to be connected together. Each switch contains an ASIC and up to eight optical link cards. In addition, an embedded control processor allows for management and configuration of the switch.

2.3 Striping and Fail Over

The NI can use striping to distribute traffic among different paths to a destination host. Striping can be done between different NI's, or between the links on one NI.

In the event of a link or NI failure, software reconfigures the system so that network

traffic flows over a different set of links, avoiding the failed component. This may result in a reduced degree of striping. Fail over is transparent to MPI applications, which will continue running without interruption after a failure, as long as one functional link remains.

2.4 Network Topologies

Clusters of two or three hosts may use a point-to-point network configuration; larger clusters require a switch. While the Sun Fire Link architecture can support up to 254 hosts, the current switch implementation is limited to 8 hosts, and switches cannot be cascaded. Nodes may be connected to multiple networks in order to permit striping and fail over. Legal topologies are shown in Figure 1.

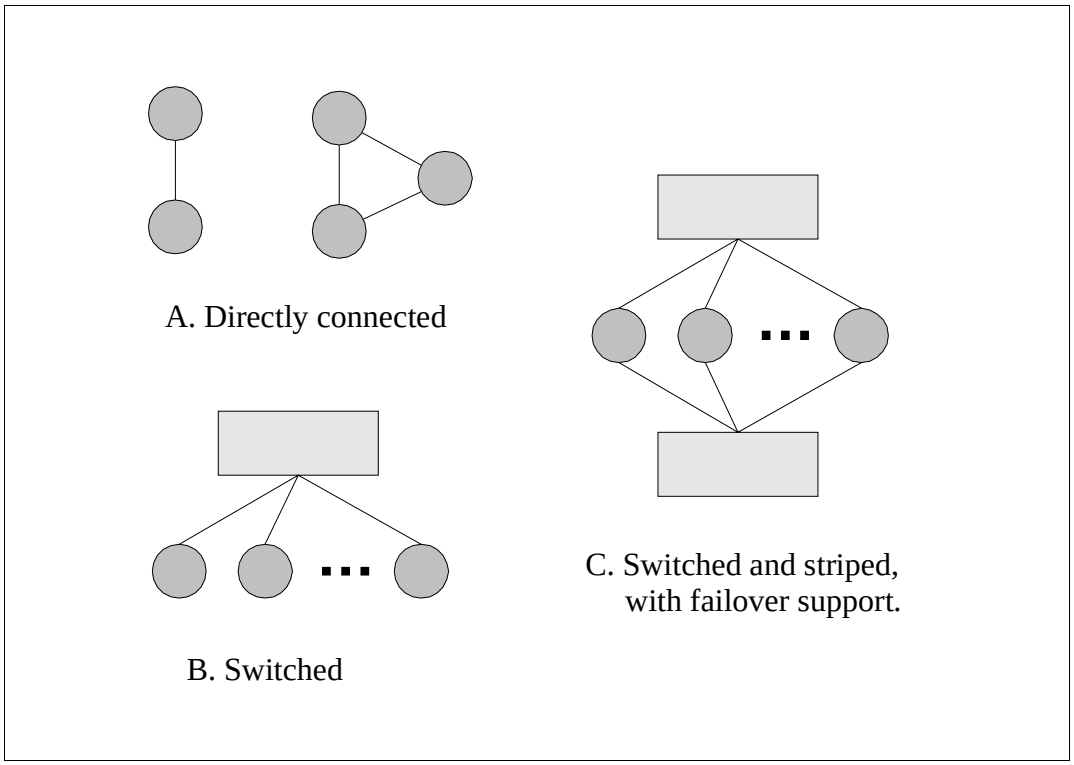

Figure 1. Topologies. Circles represent compute nodes.

2.5 Block Load/Store Access to Remote Memory

The primary feature provided by the network interface is the ability to read and write memory on another cluster host. Only memory that has been specifically made available, or exported, by the remote host may be accessed. Memory may be exported to a specified subset of the cluster's hosts in read-write or read-only mode. Software can implement higher-level access controls which restrict access to a certain user or group. Depending on the number of hosts in the cluster and the page size used, each host can export between 16 and 128 GB of memory. A host may import up to128 GB from every host in the cluster.

Remote accesses are uncached, and are provided only to full 64-byte cachelines; that

is, a read or write operation always transfers 64 bytes, and the data must be aligned on a 64-byte boundary. This means that a thread cannot use normal load and store instructions to access remote memory. Instead, it must use the UltraSPARC® processor's Block Load and Block Store instructions. These are non-privileged instructions, so data transfer may be accomplished directly by user threads as well as kernel threads. However, since the Sun compilers do not generate these instructions, most data transfer is done by calls to specialized library routines.

The NI does not include a DMA engine. However, a high degree of pipelining both within the UltraSPARC III processor and the NI itself results in very high write throughput; a single processor can utilize a majority of a link's bandwidth.

2.6 Remote Interrupts

The network interface also provides the ability to send interrupts to processors on other cluster hosts.

Remote interrupts are initiated with Block Store instructions, and are treated as writes to remote memory by the sending NI. Translation tables on the receiving NI determine whether writes to a particular cluster address range will be treated as writes to memory, or as interrupts. For interrupts, the first few bytes of the 64-byte data packet are overwritten with information identifying the sender, and the data is then delivered to a processor specified by the tables.

Since the interrupt decision is made by the receiving NI, and the interrupt data is tagged to reliably identify the sender, the ability to send remote interrupts can safely be given to user threads.

2.7 Exception Handling

Remote reads, writes, or interrupts may sometimes fail. This could be due to a permission problem (for instance, an attempt to access remote memory which has not been made available to the requesting machine), or to a communication error (for instance, data errors on an optical link). A cluster program must be able to detect and recover from these sorts of errors.

An NI informs a requesting processor of a communications error by use of the Cluster Error Status Register (CESR). When an error occurs, the CESR is set to a code which denotes the type of fault detected.

Each NI contains multiple CESRs, one for each processor in the cluster host. Each CESR reflects only the errors incurred by transactions from the corresponding processor. Thus, when a process reads its CESR, it sees only errors for which it is responsible.

In order to provide each process with its own virtual CESR, the per-processor CESR is saved and restored on each context switch. Note that all CESRs are accessed with the same physical address; the data returned differs, depending on the requesting processor. Because of this, no register mapping changes are necessary when a process moves to a new processor.

2.8 Software Interfaces

Low level access to Sun Fire Link transport from user space is provided by the

RSMAPI library [8]. An equivalent library called RSMPI exists for kernel space use. RSMAPI is a transport independent API for remote data access that can be implemented on any interconnect that supports remote shared memory. Transport specific details are confined to a transport specific plug-in library. We have written plug-in libraries for the Sun Fire Link interconnect and for SCI [9].

RSMAPI provides functions to retrieve topology information, to export and import memory segments, to put and get remote data, and to perform memory barrier operations. Data can be written to or read from an imported memory segment using either the put and get functional interfaces, or appropriate processor Load and Store instructions. The memory barrier operation guarantees that previous stores are committed to remote memory, and reports any transport errors which have occurred since the previous barrier.

2.9 Comparison to Other Interconnects

The ability to expose local memory in a network address space is not new. It has been supported in hardware by a number of interconnects over the years, including SCI, Memory Channel [10], QsNET [11], and more recently the InfiniBand™ architecture [12]. Of these, a Sun Fire Link adapter is most similar to an SCI adapter in that remote operations are driven by programmed I/O (PIO). Both adapters support read and write transactions, whereas Memory Channel only supports remote writes. However, a Sun Fire Link adapter implements a more robust memory mapping and protection model, and can export much more memory, than the implementation of SCI that is available on SPARC® platforms.

PIO based adapters rely on the processor to initiate a remote operation by issuing a memory transaction on the system backplane to a region of the I/O address space. The adapter intercepts the transaction and forwards it to be re-issued on the remote node. In contrast, QsNET and InfiniBand adapters are based on DMA engines. To perform a remote operation, an application builds a descriptor representing the operation and posts it to a region of the device's memory that is mapped to user space. The device then accesses the referenced buffer using the DMA engine. Because of the extra backplane transactions required to build and post a descriptor, DMA transfers will usually have higher remote latencies for small data transfers than PIO transfers.

The InfiniBand architecture offers a rich set of capabilities, including send, receive, RDMA, atomic operations, connected and datagram transport, asynchronous completion notification, alternate path fail over, and multicast. It also promises interoperability between different implementations, and will provide high bandwidths with its 4x and 12x link technologies. For these reasons, we expect it to become the interconnect of choice in the future for clustering applications. However, at this time InfiniBand technology is not widely available in a complete hardware and software stack, and existing host channel adapters are PCI based, which increases latency and limits bandwidth.

3. Sun MPI Architecture

The overall Sun MPI architecture is shown in Figure 2. Interconnect-specific transport protocols are found in the bottom layer. To support the Sun Fire Link interconnect we have written a new protocol module based on RSM and implemented on top of RSMAPI.

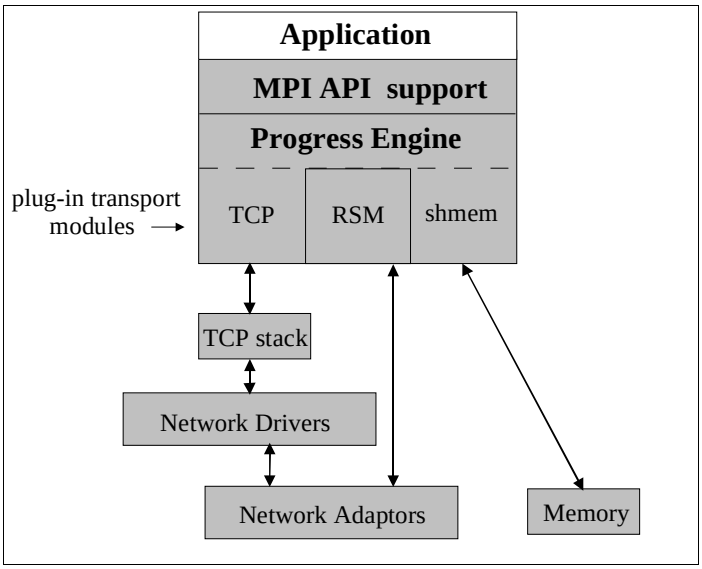

Figure 2. Sun MPI library architecture

Based on the interfaces available at run time, the library picks the most efficient protocol to use between each pair of processes. This will be the shared memory protocol for processes co-located on an SMP or the RSM protocol for distant processes.

The RSM module implements a memory-to-memory messaging paradigm in which the Sun MPI library has total control over RSM buffer management. In contrast, many MPI implementations that support kernel bypass messaging are layered on top of a send-receive primitive exported by the NI. Our scheme has several advantages, which shall be described in the following sections.

3.1 Buffer Management

As shown in Figure 3, each MPI process owns a memory area on each destination host, which may be used to send data to *any* of the receivers on that host. Note that we do not have a separate area for each potential receiver; the memory requirements on each host would grow as the square of the total number of processes, with a large constant factor, which is untenable for large jobs. The memory area physically exists on the destination host, and the sender accesses it by writing to mapped addresses in the virtual address space of the source host.

Each memory area (a single gray rectangle) in Figure 3 is managed by a single sending process, and is shown in expanded detail in Figure 4. The memory area is composed of postbox queues that point to messages in a buffer pool. A postbox is 64 bytes, which by design is the size of a single cache line on an UltraSPARC processor. There are multiple postboxes per queue, and one queue per destination rank. The sender allocates space from the pool in multiples of small or large block sizes (currently 2K and 4K bytes), which avoids fragmentation of the pool into inefficiently small pieces. A sender has exclusive control over its pool, so no expensive mutual exclusion primitives are required for allocation or deallocation operations. The pool is flexible in that a sender may:

- allocate most of the pool for a single large message to a single destination
- divide the pool for multiple smaller messages to a single destination

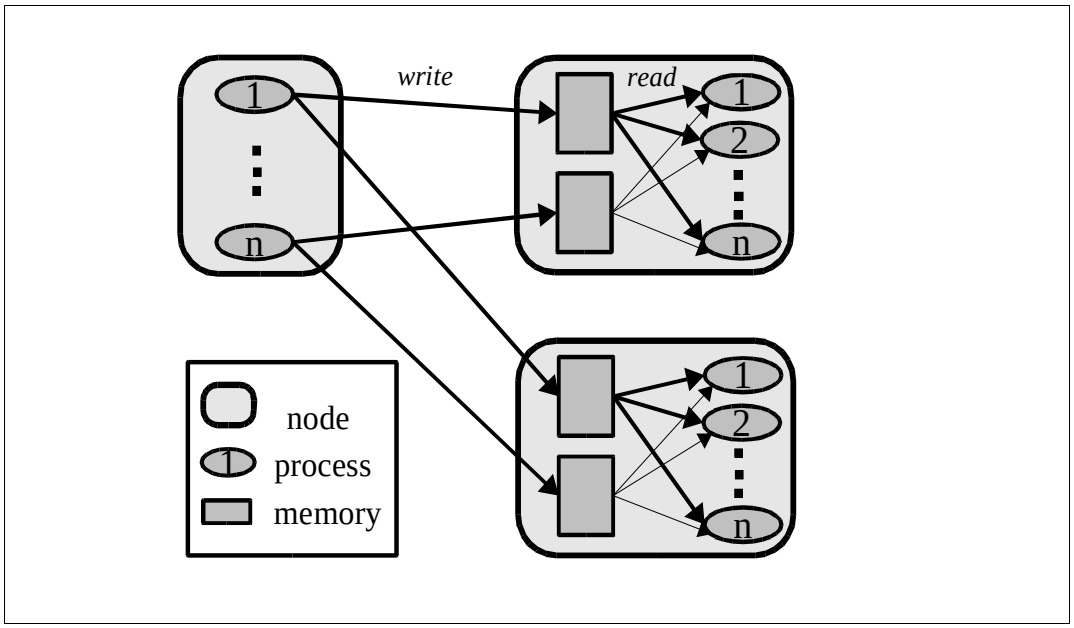

Figure 3. Remote buffer placement

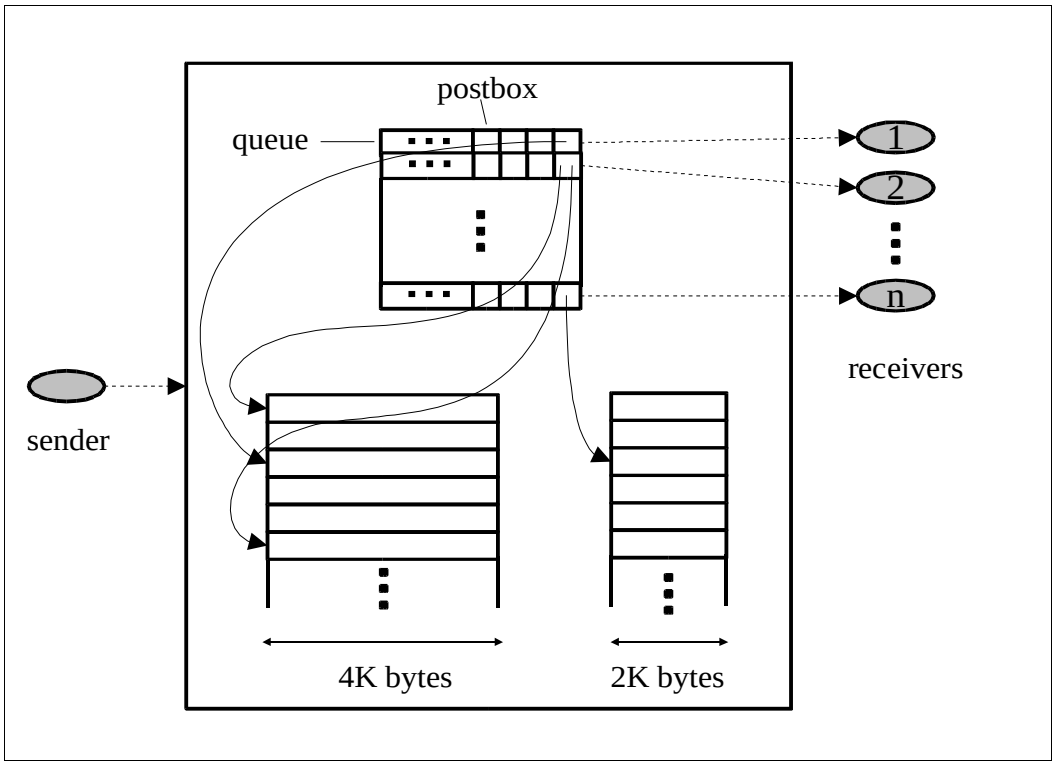

Figure 4. Structure of message buffers

- divide the pool for multiple messages to multiple destinations

The number of postboxes in a queue, the size of the buffer pool, and the block sizes are all tunable parameters.

3.2 Transfer Protocols

To transfer a large message, the sender:

- allocates blocks from the pool
- copies the posted message into the blocks
- issues a memory barrier operation
- packs pointers to the blocks in one or more postboxes at the end of the queue
- sets a flag in the postbox telling the receiver that a new message is present

The memory barrier operation ensures that the flag does not arrive before the data buffer is complete. The receiver will notice the set flag the next time it polls the postbox at the head of its queue when looking for messages from this sender. After extracting the message, the receiver writes a 1-byte acknowledgment to an "ack segment" on the sender's host. This allows the sender to reclaim its buffer space.

A single message may span multiple postboxes. The receiver can extract pieces from earlier postboxes while the sender fills later postboxes, which allows full pipelining between the sender and receiver and doubles the transfer rate.

The memory barrier operation is supported by the Sun Fire Link hardware, but is still relatively expensive because it involves a round trip message. To avoid a barrier and achieve lower latency for short messages, we use an alternate protocol, in which message data is stored directly in the postboxes. The receiver acknowledges receipt of each postbox separately but asynchronously. This method is more efficient for messages that are smaller than a few thousand bytes. By design, the protocols described here are based on remote writes, which are more efficient than remote reads on our architecture, and many others as well.

3.3 Connection Management

The connection described in the transfer protocol above is half duplex, in that the receiver can only return ACKs to the sender. If the receiver wants to send MPI messages, it creates its own set of queues and buffers on the destination host. Further, these half-duplex connections are not established until the first message is sent to a particular destination. This minimizes application startup time and avoids wasting resources on connections that will never be used.

When multiple Sun Fire Link interfaces are available we arrange that each interface export different parts of the buffer pool. By breaking messages into pieces and writing them to separate pools, we increase the transfer rate through a software striping technique that complements the hardware striping done by the NI.

3.4 Memory Efficiency

Our protocols are designed to minimize latency and maximize bandwidth, but just as

importantly they are designed to optimize the distribution and use of buffering resources. The half duplex, deferred connection, and shared receiver pool strategies avoid wasting resources where they are not needed, and the saved resources can be applied at destinations that will benefit from an increased allocation.

Large buffer allocations increase application performance by allowing senders to deliver their messages and continue, regardless of whether the receiver is ready to accept them. This is a significant effect, because it is rare in real applications for senders and receivers to be synchronized perfectly.

Efficient use of buffer space also allows our MPI implementation to scale to thousands of ranks using only a small fraction of the memory that can be exported through the NI.

4. Performance

In this section we present MPI-based performance data that is meant to characterize the effectiveness of our MPI implementation in concert with the Sun Fire Link interconnect.

4.1 Configuration

The configuration for all experiments consists of an 8-node cluster of Sun Fire servers, each containing 24 UltraSPARC III processors running at 900 MHz, for a total of 192 processors. Each server has 2 network adapters containing 2 links per adapter. This is the favored network-adaptor configuration because of its high fault tolerance. The theoretical peak off-node bandwidth for each server in this configuration is 4.8 GB/s.

Most of these experiments use only two nodes of the cluster to demonstrate the performance of the network at an endpoint. Some larger tests demonstrating scalability for greater numbers of processors and nodes are also shown.

4.2 MPI Inter-node Performance

The MPI inter-node ping-pong latency for zero length messages on this configuration is 3.7 microseconds, measured as half of the round-trip time. If we bypass MPI and instead use RSMAPI and Block Store instructions, we can achieve a ping-pong latency as low as 1.7 microseconds. Thus, the MPI layer adds a few microseconds of overhead to the low level ping-pong latency.

Figure 5 shows the aggregate bandwidth achieved with a standard MPI ping-pong test for varying numbers of ping-pong pairs, which are spread across two nodes such that every process has a peer on the other node. The bandwidth for each pair is measured in the usual manner as the time required to send the ping then receive the pong, divided by the sum of the two message sizes. The results are summed and plotted on the y axis. The np value represents the total number of processors used. The lower curve shows that a single MPI_Send operation can achieve an end-to-end transfer rate of 792 MB/s, where message data starts in the cache of the sending processor, and is delivered to both main memory and the cache of the receiving processor. The upper curve shows that the MPI application can generate over 2900 MB/s of inter-node bandwidth, which is 60% of the hardware peak. Notice the inflection point in all of these curves at message sizes of 2000 bytes. This point nicely shows the transition between the short and long message

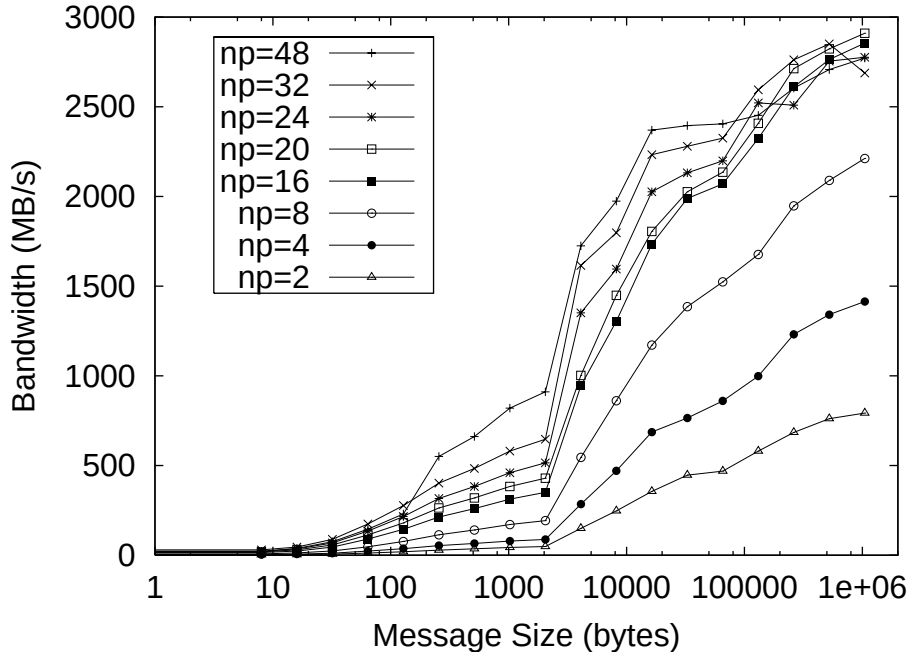

Figure 5. Aggregate ping-pong bandwidth

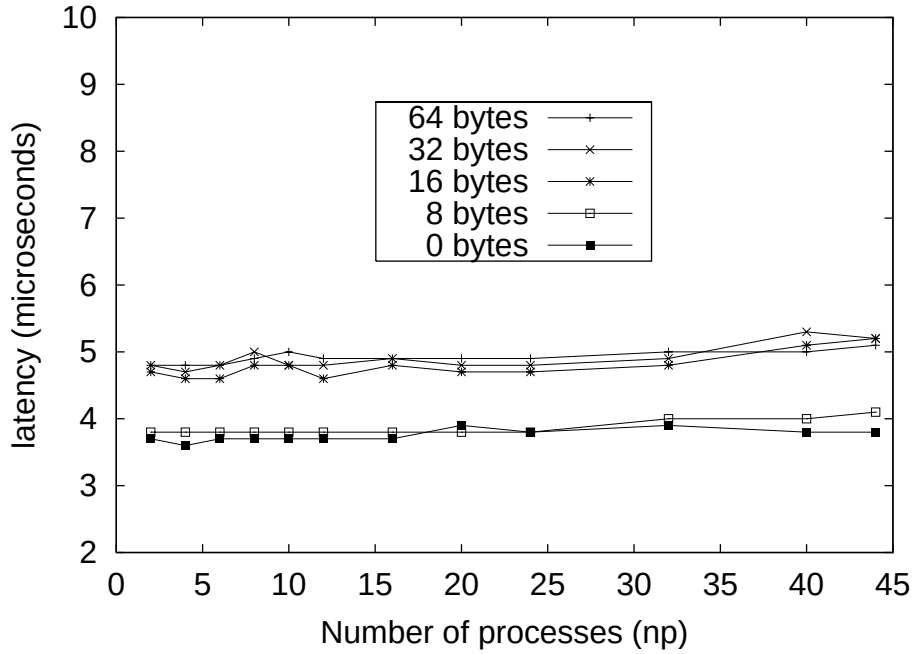

Figure 6. MPI latency under load

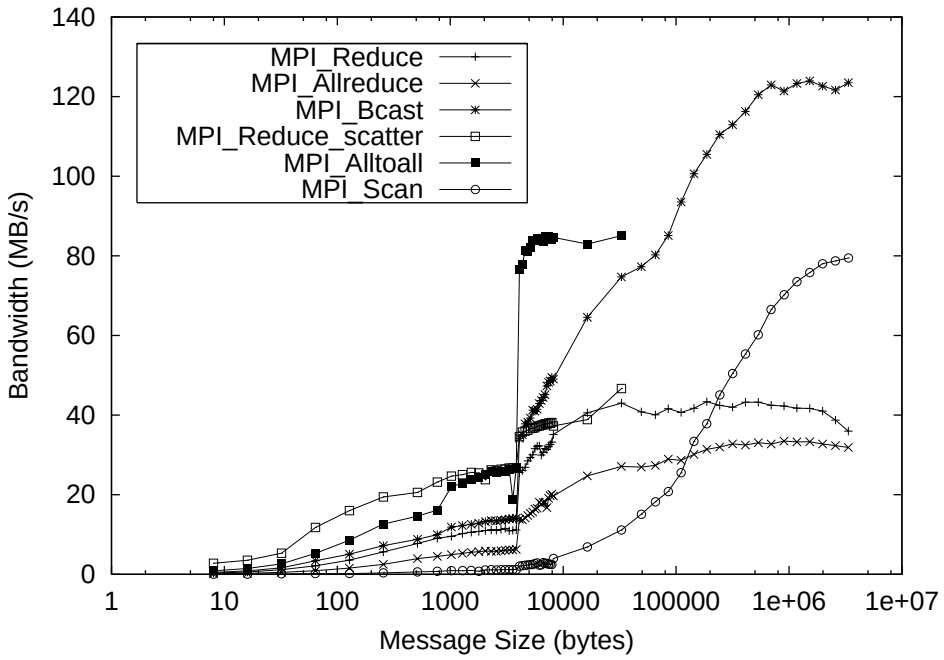

Figure 7. MPI collective performance for np=92 across 4 hosts

protocols.

Figure 6 shows ping-pong latencies for a few small message sizes. For each curve, the message size is held constant while the number of ping-pong pairs is increased, and the latency is plotted. The flatness of the curves shows that our short message protocol delivers a robust low latency that is not sensitive to load; it remains low even when many pairs are using the interconnect.

Figure 7 shows the performance of several MPI collectives at a variety of message sizes. The bandwidth is the throughput seen by the root process, or by any single process in the case of MPI_Alltoall. These tests are run at np=92, with 23 ranks on each of 4 nodes. The MPI_Alltoall performance reaches 85 MB/s, which means that all endpoints together are pushing 7820 MB/s of application data to each other. The vertical discontinuity in the graph is caused by the transition from the short to long message protocol, and indicates room for improvement in tuning this transition.

4.3 SMP Performance Comparison

This section shows a series of experiments where a parallel program is first run on a single node and is then run on 2 nodes of the cluster. The total number of processes is the same in both cases. Table 1 shows the basic MPI ping-pong metrics for a single pair of processes. The on-node and off-node bandwidths are comparable, and the latencies differ

	on-node	***off-node***
latency (μs)	1.5	3.7
bandwidth (MB/s)	806	867

Table 1. MPI ping-pong metrics, single pair

operation	*on-node*	*off-node*	*ratio*
MPI_Gather	627	626	1.00
MPI_Gatherv	718	718	1.00
MPI_Scatter	738	760	1.03
MPI_Scatterv	784	810	1.03
MPI_Bcast	183	182	0.99
MPI_Allgather	151	156	1.03
MPI_Allgatherv	171	179	1.05
MPI_Alltoall	280	295	1.05
MPI_Alltoallv	325	324	1.00
MPI_Reduce	41	40	0.98
MPI_Reduce_scatter	41	42	1.02
MPI_Allreduce	34	33	0.97
MPI_Scan	87	95	1.09

Table 2. MPI collective bandwidth, MB/s.

by a ratio of 2.5:1, which is quite good.

Table 2 compares the on and off-node performance of MPI collective operations using large messages. The job has 16 MPI processes in both cases. This is a tough comparison, because the collectives are highly optimized for the single node case [13]. Nevertheless, the off-node performance is often better than the on-node performance, as shown by the cases where the ratio is greater than 1. This is a strong indicator that the Sun Fire Link interconnect can deliver SMP-like performance.

benchmark	*time ratio*
BT	1.08
CG	0.96
EP	1.00
FT	1.05
IS	0.94
LU	0.96
MG	1.20
SP	1.01

Table 3. NPB performance comparisons

Table 3 shows the ratio of on-node to off-node execution times for NPB class A benchmarks using 16 MPI processes. Again, values greater than 1 indicate that the job actually ran faster when spread across two hosts.

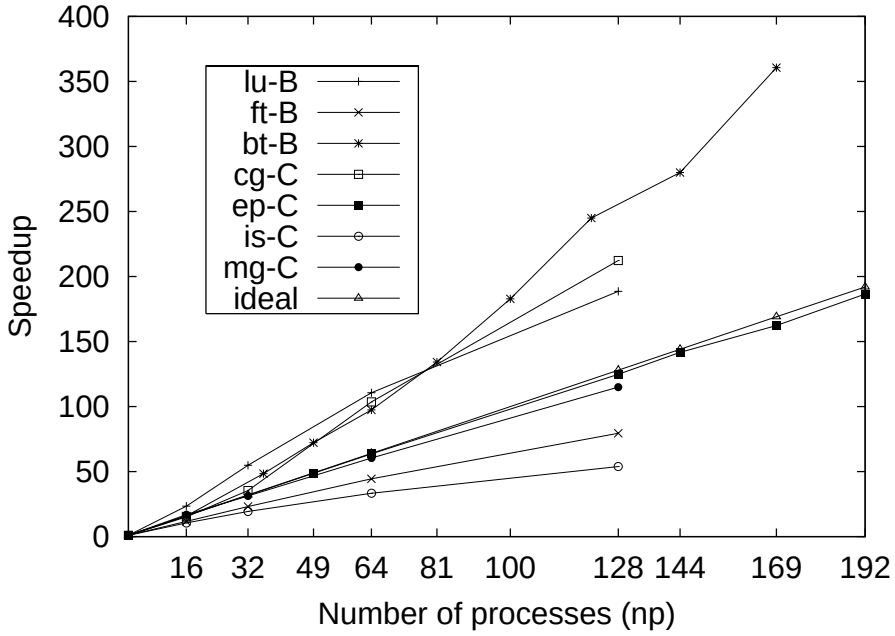

Figure 8. NPB scaling.

Figure 8 shows the relative speedup obtained for NPB class B and C benchmarks as more nodes and processors are added. Speedups are close to or exceed the ideal in all cases, which shows that our 2 node results continue to scale on larger systems.

Overall, the results in this section consistently show that programs running on our cluster can achieve SMP-like performance.

5. Futures

The high performance of the new interconnect dramatically alters the performance balance between on-node and off-node operations, which affects the transition points in our multi-protocol MPI collective algorithms. We may spend some time tuning these algorithms. We also plan to implement the MPI one-sided communication API over RSM in a future project. Lastly, we expect to port our MPI memory-to-memory messaging implementation to InfiniBand interconnects, and to take advantage of InfiniBand specific features to achieve further improvements in performance and efficiency.

6. Conclusion

We have developed new hardware and software interconnect components that not only deliver high performance in absolute terms, but more importantly provide the enabling technology for building capability clusters using large SMP servers as building blocks. Hardware striping enables high performance, transparent fail over enables high reliability, and hardware support for general-purpose remote shared memory provides flexibility to higher level software libraries such as MPI.

The Sun MPI library implementation is well-matched to the capabilities of the Sun Fire Link hardware. Conservative and dynamic use of remote buffering coupled with multiple transport protocols delivers the performance potential of the interconnect to the application, robustly and scalably. Together, our hardware and software implementation delivers off-node performance that exceeds the capacity of the SMP backplane in many previous generation servers.

Acknowledgments

We would like to thank Mike Johnson and Eugene Loh for gathering much of the data that appears in this paper. We would also like to thank the many engineers at Sun Microsystems who contributed to the Sun Fire Link and Sun MPI implementation. The quality of the result clearly reflects their talent and effort. This work was supported in part by the ASCI PathForward Program on Interconnect Technologies (PF-01), subcontract B338313, Lawrence Livermore National Laboratory.

References

[1] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet - A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29-36, February 1995.

[2] Pakin, Lauria, and Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. *Proceedings of Supercomputing '95*, San Diego.

[3] Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V. Packer. BEOWULF: A Parallel Workstation for Scientific Computation. *Proceedings of the International Conference on Parallel Processing '95*, August 1995.

[4] Thomas E. Anderson, David E. Culler, and David A. Patterson. A Case for Networks of Workstations: NOW. *IEEE Micro*, February 1995.

[5] Steven S. Lumetta, Alan M. Mainwaring, and David E. Culler. Multi-Protocol Active Messages on a Cluster of SMP's. *Proceedings of Supercomputing '97*, San Jose.

[6] Steven S. Lumetta and David E. Culler. Managing Concurrent Access for Shared Memory Active Messages. *IPPS/SPDP '98* , Orlando.

[7] Lisa Noordergraaf and Robert Zak. SMP System Interconnect Instrumentation for Performance Analysis. *Proceedings of Supercomputing 2002*, Baltimore.

[8] Sun Microsystems. librsm(3LIB) man page. http://docs.sun.com.

[9] IEEE Standard for Scalable Coherent Interface (SCI), IEEE Std 1596-1992.

[10] R. Gillett. Memory Channel Network for PCI. *IEEE Micro*, 16(1):12-18, February 1996.

[11] Fabrizio Petrini, Adolfy Hoisie, Wu-chun Feng, and Richard Graham. A Performance Evaluation of the Quadrics Interconnection Network, In *Workshop on Communication Architecture for Clusters 2001, International Parallel and Distributed Processing Symposium 2001*, April 23-27, 2001. San Francisco.

[12] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1, Release 1.0.a*, June 2001. http://www.infinibandta.org

[13] Steven Sistare, Rolf vandeVaart, and Eugene Loh. Optimization of MPI Collectives on Clusters of Large-Scale SMP's. *Proceedings of Supercomputing '99*, Portland.