

An Ultrahigh Performance MPI Implementation on SGI[®] ccNUMA Altix[®] Systems

Karl Feind and Kim McMahon

Silicon Graphics, Inc.

(Rec. 11 September 2005)

Abstract: The SGI[®] Message Passing Toolkit (MPT) software has implemented algorithms that provide extremely high-performance message passing on SGI Altix[®] systems based on the SGI NUMALink[™] interconnect technology. Using Linux[®] OS infrastructure and SGI XPMEM cross-host memory-mapping software, SGI MPI delivers extremely high MPI performance on shared-memory single host/SMP Altix systems as well as multihost superclusters. This paper outlines the Altix hardware features, OS features, and library software algorithms that have been developed to provide the low-latency and high-bandwidth capabilities. We present high-performance features like direct copy send/receive, collectives, and the ultralow-latency SHMEM[™] data transfer library. We include MPI benchmark results, including an MPI ping pong latency that ranges from 1.2 to 2.3 microseconds on a 512-CPU Altix system with 1.5 GHz Intel[®] Itanium[®] 2 Processors.

Key words: MPI, MPT, message passing, ccNUMA, Altix, memory access, shared memory, high bandwidth, low latency

1. INTRODUCTION

SGI ccNUMA (known as SGI NUMA) servers, including SGI Origin[®] 3000 and SGI Altix 3000 systems, provide many programming options for writing extremely scalable parallel programs. The large number of CPUs per host provide an opportunity for highly scalable programs to use SMP-like memory-sharing approaches such as OpenMP, System V shared-memory segments, and memory-mapped files. However, the majority of scalable user applications in existence are written using the MPI parallel programming model, and it is important for a vendor to provide an MPI environment that delivers on the scalability opportunities offered by the low latency, high bandwidth, and fast synchronization available on such shared-memory systems. The MPI programming model also allows these fast communication services to be available whether the system is a single large host or a supercluster of hosts connected with the NUMALink interconnect network.

SGI systems based on the NUMALink interconnect network have a unique capacity for ultralow-latency implementations of librarybased send/receive or put/get communication methods. Because the memory on all hosts in a system connected using the NUMALink interconnect can be directly accessed, simple approaches that utilize load/store and atomic memory operations (AMOs) may be used to implement ultra-low-latency communication across the entire system. The advantages of this load/store/AMO-based implementation are magnified for simpler communi-

cation approaches, such as put/get in the MPI-2 [1] and SHMEM [2-4] specifications. In put/get communication, the library call overhead is very minimal; therefore, the low communication latency made possible by the hardware maps very closely to the observed latency of the end user.

In this paper, we present the implementation of fast MPI send/receive and put/get and SHMEM put/get for Altix systems. We show an approach for capitalizing on the advantages of SGI NUMA systems for implementation of library-based message passing on large collections of Linux hosts connected in the same NUMALink interconnect network. We show that an ultrafast MPI implementation can bring the advantages of systems based on the NUMALink interconnect to portable, highly parallel MPI user applications.

2. ALTIX SYSTEM SOFTWARE OVERVIEW

User programs and libraries on an SGI Altix 3000 system have direct access to permitted memory across the whole system. As we will see, this feature allows the MPI implementation much freedom for implementation of an efficient message send algorithm. The system is administered as a supercluster of a number of Linux hosts. All hosts in the supercluster are individual partitions within a larger NUMALink interconnect domain. The Linux software distribution installed on Altix systems is augmented by SGI software that provides memory sharing within and across hosts (XPMEM [5]), optimized libraries, and other

features. The software layers related to memory access on the NUMALink interconnect are shown in Fig. 1.

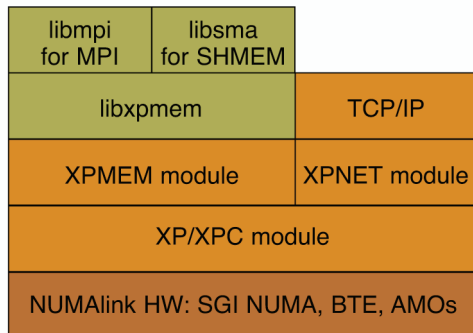


Fig. 1. Software/Hardware Stack for NUMALink Interconnect

The XPMM kernel module provides kernel call-outs in the key areas of memory allocation, deallocation, and first-touch. These capabilities support the user libraries as they attach (map) memory from throughout the system into each MPI process virtual address space. SGI MPI programs attach all memory in the static and common blocks segments, the stack segments, and the heap segments of an MPI process into the virtual address space of every other MPI process in the job. This allows free access to MPI queues and data structures, as well as user data areas passed as send and receive buffers or targeted by put or get operations.

3. MESSAGE PASSING SOFTWARE OVERVIEW

The MPT product contains at its core an efficient message queuing algorithm, which is described in section 4. The MPI message queue and algorithms for high bandwidth provide fast communication for MPI send/receive operations and collective communication functions defined by MPI-1. These are the most crucial functions for existing portable MPI applications, which have predominantly adopted the use of MPI-1 communication and, to a lesser extent, MPI-2 and SHMEM functions to implement the needed communication in parallel programs. The ultralow latency provided by MPT results in measurable scalability improvements in MPI applications like the CFD application FLUENT[®] [6].

In addition to providing high performance MPI, the MPT package includes functions that extend MPI and provide convenient user access to the global shared-memory capabilities of the system. These capabilities include the SHMEM message passing API and global pointer interfaces including `shmem_ptr()` and `MPI_SGI_globalptr()` [7]. These global shared-memory extensions may be used as the

primary communication method in a parallel application. Alternatively, they may be used to augment MPI in portions of the application that need extremely low communication latency or as an enabler of advanced dynamic parallel load-balancing algorithms that can enable an MPI program to scale to larger numbers of processors.

4. THE MPI SEND/RECEIVE ALGORITHM

The data structures involved in an MPI send/receive operation are as follows. Every MPI process has its own outgoing buffer pool and incoming message header queue. The buffer pool stores copies of the payload for buffered sends. The message header queue is a circular queue of entries 128 bytes in size, which is the cache-line size. Message header queue entries hold information about the MPI messages – type, size, tag, and pointers to the buffered data. Using shared-memory programming techniques, message headers are efficiently enqueued into the message header queue during the processing of an MPI message send operation. The buffered payload data is pulled into the user receive buffer by the MPI message receive operation.

There are two variations of the above algorithm that are worthy of note. The first is for short messages less than or equal to 64 bytes in length. Short message payloads are placed in the message header instead of the outgoing message buffer pool. The second variation, termed direct copy data transfer, is an optimized way to transfer large messages. The receiving process copies the message payload directly from the user's send buffer, thereby obviating the need to buffer the message in the message buffer pool. The direct copy data transfer method is chosen automatically by the MPI library for messages larger than a user-tunable threshold when processing `MPI_Isend`, `MPI_Sendrecv`, and most MPI collective communication functions.

An important aspect of the buffer pool scheme is that all of memory is accessible for push or pull operations. Hence, a single buffer pool can hold all outgoing messages from a process, and all remote processes can pull the data out to deliver the message. This permits linear scaling of the buffer pool space needed as the number of MPI processes increases, and the buffer pool space is not at all dependent on the number of processes per Linux host. Similarly, the generalized push capability enables the use of a single message header queue per process where all senders deposit incoming message headers.

5. THE MPI SEND/RECEIVE ALGORITHM ON MULTI-CACHE COHERENCY DOMAIN SYSTEMS

The memory controller ASIC developed by SGI and deployed in Altix systems is known as SHUB. The SHUB

extends ccNUMA capabilities to 512 CPUs. SGI plans to scale ccNUMA features and performance to larger systems in future generations of the SHUB ASIC, but in the current generation the message passing libraries accommodate the transition from hardware-managed cache coherency to software-managed cache coherency in global memory accesses. For MPI jobs larger than 512 CPUs or those that span multiple 512-CPU cache coherency domains, the message passing library software must assist the hardware in maintaining cache coherency. In MPI jobs that span multiple cache coherency

domains, the MPI-1 send/receive and collective communication functions are supported, but the MPI-2 and SHMEM put/get operations are not currently supported.

6. POINT-TO-POINT COMMUNICATION PERFORMANCE

Figure 2 shows the latency measured for MPI send/-receive, MPI_get, SHMEM get, and shared-memory refer-

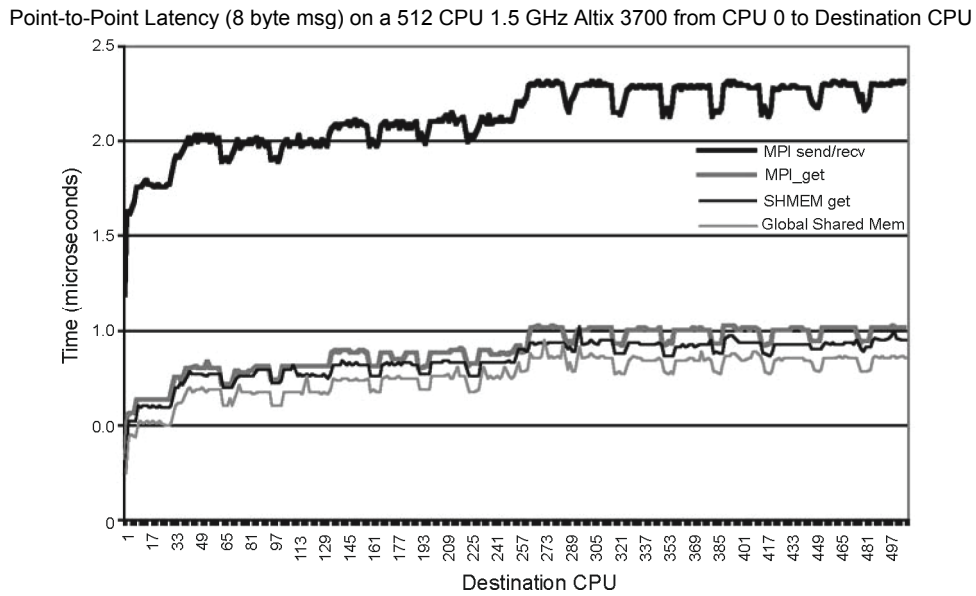


Fig. 2. MPT Communication Latency

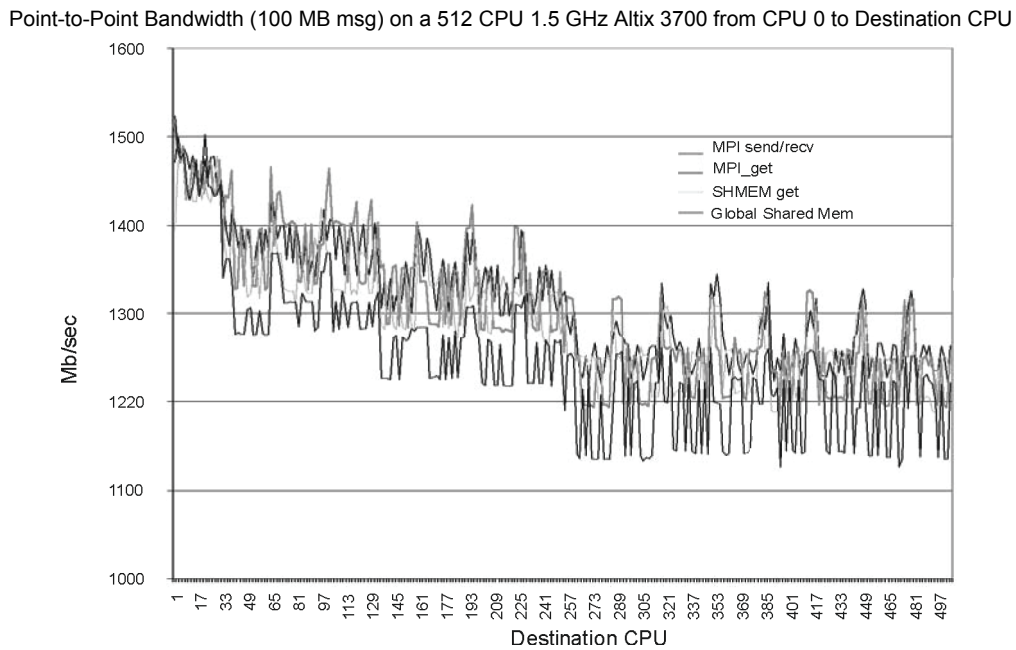


Fig. 3. MPT Communication Bandwidth

ences. The send/receive measurements are half the time to do 8 byte ping pong message exchanges. The get and shared-memory measurements are for remote cache-miss memory references. The best latency is seen at the point where the sender and receiver are on the same memory node. In this case, bus snooping turns the cache lines around very rapidly. For off-node cases, the latency is incrementally higher when the number of network hops and the length of NUMALink cables increase.

The MPI and SHMEM get latency line has a similar contour as the MPI send/receive latency line, but two significant effects can be seen. First of all, the latency for a get operation is about 1.3 microseconds less than the MPI send/receive latency. This is the result of lower library overhead associated with implementing the semantics of a get operation compared with the semantics of MPI send/receive. A second effect is that the latency increments for a get operation are even lower than that for send/receive as distance and router hop counts increase. This is because a get operation needs to send fewer hardware protocol-level messages between the source and destination than for a send/receive data transfer.

Figure 3 shows the point-to-point peak bandwidth achieved for 100 MB messages. The SGI message passing libraries include an optimized bulk data copy function, `_fastbcopy()`, which maximizes data transfer bandwidth for block copies when the source buffer is not in cache. The message passing libraries use this function to transfer data inside the `MPI_recv`, `MPI_get`, and `SHMEM_get` functions. Peak bandwidth measurements for large message transfers are equal for all the measured data transfer methods across the entire ccNUMA domain.

7. CONCLUSIONS

The SGI Altix 3000 system provides global access to memory distributed across multiple hosts and multiple memory nodes within hosts. This system's hardware architecture offers memory sharing capabilities that permits fast communication and synchronization for parallel programming. For systems up to 512 processors, the cache-coherent memory and system software layers make the access to shareable memory easy and enable implementation of highly scalable MPI communication schemes, as well as a number of extensions like SHMEM programming and the use of global pointers. The implementation of MPT message passing and other communication features is scalable and fast from the perspectives of latency, point-to-point bandwidth, and synchronization.

References

- [1] <http://www.mpi-forum.org/docs/docs.html>
- [2] `intro_shmem` man page at <http://docs.sgi.com>
- [3] <http://www.shmem.org>
- [4] R. Barriuso and A. Knies, SHMEM User's Guide for Fortran, Cray Research Inc. (June 1994)
- [5] Michael Woodacre, Derek Robb, Dean Roe and Karl Feind, The SGI Altix 3000 Global Shared-Memory Architecture, <http://www.sgi.com/pdfs/3474.pdf>
- [6] Exploiting the Scalability and Power of FLUENT: The SGI Message Passing Toolkit on the SGI Altix High-Performance Computing Platform powered by the Intel Itanium 2 Processor; Whitepaper by Intel, Fluent, and SGI; <http://www.sgi.com/pdfs/3807.pdf>
- [7] `shmem_ptr()` and `MPI_SGI_globalptr()` man pages at <http://docs.sgi.com>

This documentation, in electronic format, comprises software developed at private expense; if acquired under an agreement with the USA government or any contractor thereto, it is acquired as "commercial computer software" subject to the provisions of its applicable license terms and conditions, as specified in (a) 48 CFR 12.212 of the FAR; or, if acquired for Department of Defense units, (b) 48 CFR 227-7202 of the DoD FAR Supplement; or sections succeeding thereto. Contractor/manufacturer is SILICON GRAPHICS, INC., 1200Amphitheatre Parkway, Mountain View, CA 94043-1351.

Silicon Graphics, SGI, IRIX, and the SGI logo are registered trademarks of Silicon Graphics, Inc., in the United States and/or other countries worldwide, used with the permission of Silicon Graphics. All other trademarks and copyrights are owned by their respective owners.