WHITE PAPER

# A Hardware-Accelerated MPI Implementation on SGI® UV™ Systems

January, 2012


## Abstract

The SGI UV system has a rich set of hardware features that enable scalable programming models to be implemented with high efficiency and performance. In particular, the popular Message Passing Interface (MPI) programming model has been implemented to utilize the MPI hardware acceleration features included in the UV system. This paper describes the hardware-accelerated features of SGI UV MPI and shows the performance gains achieved by their use.


.

TABLE OF CONTENTS

# 1.0 Introduction

One of the most interesting aspects of the SGI UV Coherent Shared Memory Processor (SMP) system is the size of the system—up to 2,560 cores—that can exist within a single Linux® OS instance. Such a “huge PC” provides many potential parallel programming strategies. Techniques used on SMP systems like OpenMP and threaded access to shared memory segments are every bit as useful on scale-up UV systems as the popular MPI parallel programming model or other cluster-friendly programming models such as SHMEM™, UPC and other PGAS languages. Clearly, the UV system hardware supports much flexibility in the way of parallel programming.

Nevertheless, MPI is the predominant scalable parallel programming model in the technical compute industry, and SGI UV hardware and software have been engineered to meet the demanding performance needs of MPI users. MPI is crucial to supporting the most scalable applications on the largest scale-out multi-partition UV systems, where many partitions are connected via NUMAlink® 5 interconnect and the system size can grow up to 256K processor cores. Such systems are clusters of large SMPs, and a natural way to program a parallel application to run across the system is through the use of MPI.

# 2.0 SGI UV MPI Software Stack

The SGI MPI software stack includes a number of software components. These software items are availablefrom SGI via the SGI MPI product bundle. A list of the most significant components follows

- **MPI.** Libraries and commands to build and run MPI programs.
- **XPMEM**. This library and kernel module provides shared memory mapping for processor access and global memory mapping for references by the UV MPI Offload Engine (MOE) component known as the Global Reference Unit (GRU).
- **GRU Development Kit.** These libraries provide the API to directly control the global memory reference and MOE operations supported by the UV Hub that implements MPI messages, DMA and synchronization.
- **NUMA tools.** The dplace and omplace commands pin processes and threads in MPI programs and MPI/OpenMP hybrid programs onto CPUs. The dlook command can display a process address space complete with NUMA placement information.
- **PerfBoost.** This tool allows you to run applications that have been compiled for many popular MPI implementations with UV acceleration features activated.
- **Perfcatcher.** A runtime profiling tool that reports on MPI and SHMEM function calls made.
- **MPIInside.** An advanced MPI profiling and performance visualization tool optimized to use run-time state instead of excessively large trace files.

## 3.0 Dual Transport Methods

Because the scale-up SGI UV is a single SMP, MPI message queues and synchronization constructs can be implemented by memory reference via processor mapping of memory throughout the system. This approach is generally taken in the industry for best MPI performance within SMPs in general, and SGI MPI takes advantage of this technique as well. The SGI implementation has evolved and improved as it moved from one large SGI distributed memory SMP system to another over the years. Distributed memory support started with the SGI Origin® line of computer systems, continued in the SGI Altix® 3700 and 4700 lines, and most recently has been provided in the SGI UV. Rich and robust handling of NUMA architectures has come out of this process of continued improvements.

One major hardware feature sets UV apart from earlier SGI NUMA computer systems, and opens the door to MPI acceleration and offload as well as efficient PGAS programming model support—MOE. The MOE provides MPI message queues, innovative synchronization primitives and advanced RDMA capabilities such as strided and indexed global memory updates. The MOE provides a way to optimize MPI within a UV system, whether it is running as a single-instance machine or as a larger multi-partition (NUMAlink cluster) system. SGI MPI uses run-time heuristics in point-to-point and collective communication to choose the appropriate communication mode, either shared memory or MOE-based.

Within a single SGI UV SMP, all MPI process pairs can use either shared memory or MOE transport. Nearby nodes or in-cache messages are best suited for shared memory message passing because the latency is extremely low using this communication method. More distant nodes or messages in the 400KB to 4MB range are good fits for MOE transport. The incremental latency per interconnect hop is much lower with MOE transport than traditional shared memory. Figure 1 shows a profile of message latency across a 2,048-core UV system for the two transport methods. On such a system, the maximum distance between any two sockets is five NUMAlink interconnect fabric hops.

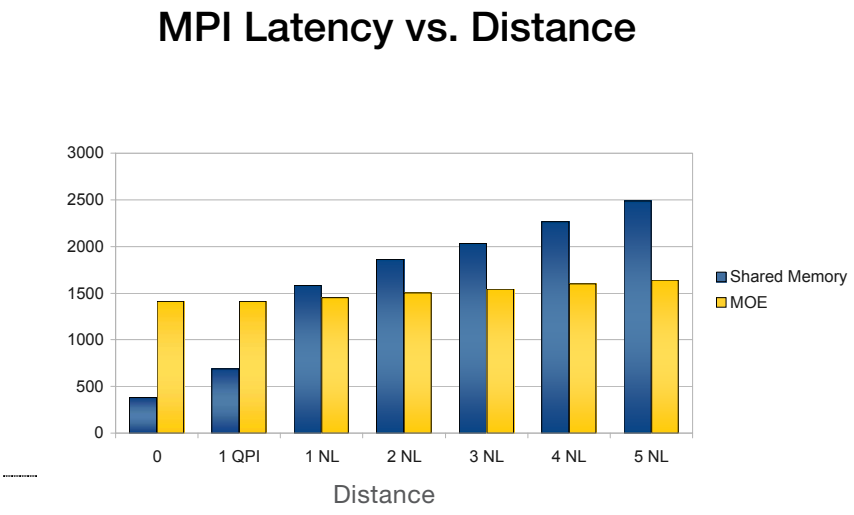


Figure 1: MPI Latency on an SGI UV System for Shared Memory and MOE Transport vs. Distance

## 4.0 SGI UV Hardware Accelerates MPI Collectives

The SGI UV MOE implements atomic memory operations in conjunction with a hardware multicast facility that helps to accelerate MPI_Barrier, MPI_Bcast, and MPI_Allreduce. All of these collectives benefit from the multicast feature, and MPI_Barrier further benefits from the way the multicast feature is triggered by certain barrier counter variable updates.

The acceleration of these collectives is a compelling advantage for applications that run with large process counts. As the process count increases, the time spent in these collectives can grow while the overall time for the application decreases. The result is a greater dependency on performance of these collectives. Figure 2 shows the performance of these collectives on UV with and without the hardware acceleration.

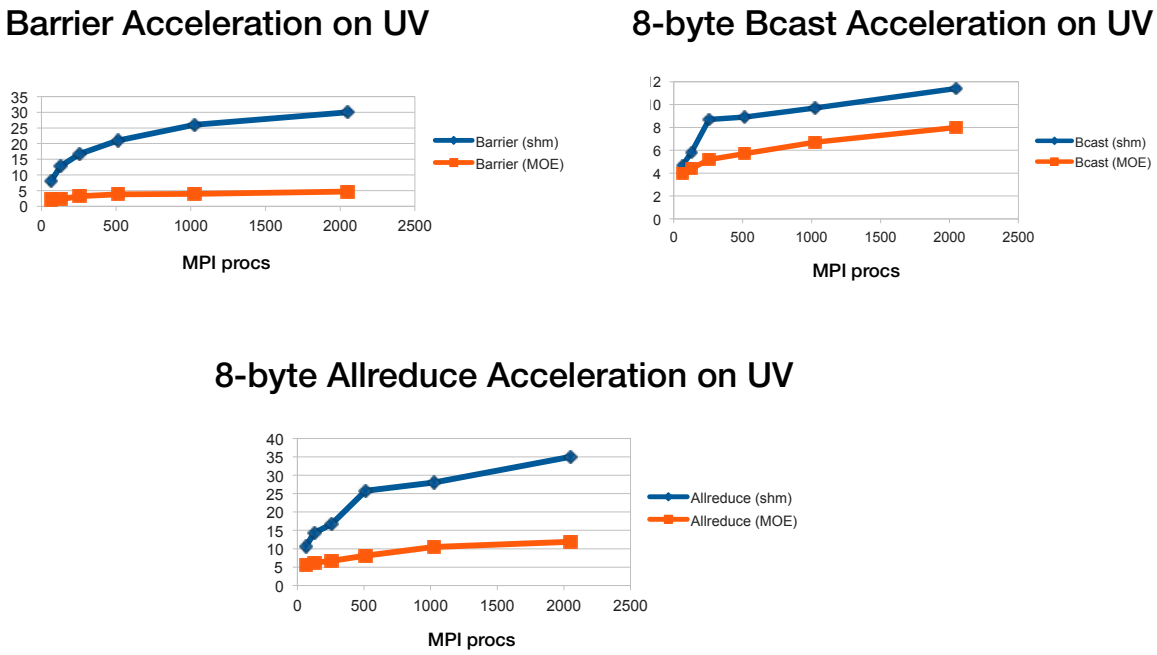


Figure 2: Accelerated MPI Collectives on SGI UV.

## 5.0 Conclusion

The SGI UV system has been designed to accelerate MPI point-to-point and collective communication. The MPI acceleration is performed by the UV MPI Offload Engine through hardware-implemented message queues, innovative hardware synchronization primitives and hardware multicast. The UV MPI implementation achieves a very flat MPI latency curve vs. distance, and industry-leading performance on some commonly used MPI collective communication operations.

**Global Sales and Support**: sgi.com/global