Optimizing Cray MPI and Cray SHMEM for Current and Next Generation Cray-XC Supercomputers

K. Kandalla, D. Knaak, K. McMahon, N. Radcliffe and M. Pagel
Cray Inc.

{kkandalla, knaak, kmcmahon, nradcliff, pags}@cray.com

***Abstract*—Modern compute architectures such as the Intel Many Integrated Core (MIC) and the NVIDIA GPUs are shaping the landscape of supercomputing systems. Current generation interconnect technologies, such as the Cray Aries, are further fueling the design and development of extreme scale systems. Message Passing Interface (MPI) and SHMEM programming models offer high performance, portability and high programmer productivity. Owing to these factors, they are strongly entrenched in the field of High Performance Computing. However, it is critical to carefully optimize communication libraries on emerging computing and networking architectures to facilitate the development of next generation science. Cray XC series supercomputers are some of the fastest systems in the world. Current generation XC systems are based on the latest Intel Xeon processors and the Aries high performance network. In addition, the Cray XC system enables users to accelerate the performance of their applications by leveraging either the Intel MIC, or the NVIDIA GPU architectures. In this paper, we present six major research and development thrust areas in Cray MPI and Cray SHMEM software products targeting the current and next generation Cray XC series systems. In addition, this paper also includes several key experimental results.**

I. Introduction

Current generation supercomputers offer unprecedented computational power and are facilitating the generation of high fidelity scientific data with significantly shorter turn-around times. Cray [1] is a world-wide leader in the field of High Performance Computing (HPC) and supercomputing. Over the years, Cray has established a strong track record of designing and deploying some of the largest and the fastest supercomputing systems in the world.

Current generation supercomputers are based on state-of-the-art multi-/many-core compute architectures, accelerators, and I/O architectures. In addition, modern supercomputers also rely on high performance interconnects to facilitate high bandwidth and low latency communication. Cray Aries interconnect [2] technology is based on the *Dragonfly* topology and offers excellent communication performance and allows parallel scientific applications to scale to hundreds of thousands of processes. Until recently, HPC systems were largely homogeneous and relied on dual-/quad- socket processor architectures. The adoption of new processor architectures, such as the Intel Many-Integrated Core architecture (MIC) [3] and the NVIDIA GPU [4], in mainstream HPC marks a critical inflection point in the evolution of the HPC ecosystem. It is widely envisioned that next generation HPC systems will primarily rely on these compute architectures. The Intel MIC architecture offers unprecedented compute density, with more than 50 compute cores per chip. However, each individual compute core offers significantly slower scalar speeds. Similarly, the latest NVIDIA K80 GPU offers up to 2.91 TFlops double precision performance, but requires the transfer of data between the host processors and the accelerators, across the network. In summary, the MIC and GPU architectures offer increased computational capabilities, but also pose new challenges. On such systems, the design and development of application and system software stacks need careful consideration in order to achieve high performance and extreme scaling.

Message Passing Interface (MPI) [5] and Partitioned Global Address Space (PGAS) [6] are two of the most common programming models used to develop parallel applications. MPI and PGAS offer primitives to implement various communication, synchronization and Input/Output operations. Parallel scientific applications rely on these primitives to exchange data, perform file I/O operations, and to manage the computational work-load across the individual compute tasks. As parallel applications are scaled out to utilize hundreds of thousands of processes, system software stacks that implement MPI and PGAS need to be designed in a highly optimized and scalable manner. Essentially, software stacks that implement the MPI and PGAS models on next generation systems must offer low latency, high bandwidth communication along with excellent scalability, computation/communication overlap and I/O performance. Considering the rapid rate of adoption of MIC and GPU architectures in mainstream HPC, unless communication libraries are optimized to reduce the cost of moving data between these compute devices across the interconnect, parallel applications will not be able to scale efficiently on next generation supercomputers.

In this paper, we take a closer look at some of the latest performance optimizations and features in Cray MPI and Cray SHMEM [7] software stacks. Cray MPI and Cray SHMEM software stacks are proprietary implementations of the MPI and SHMEM specifications, optimized for the Cray Aries [2] interconnect. Specifically, we discuss some of the latest optimizations and new features in these software stacks targeting the current generation processor architectures, and Cray interconnects. This paper presents some of our recent efforts to improve the performance of I/O intensive scientific applications. This paper also describes our approach to improve the performance of multi-threaded MPI communication. Finally, this paper includes a brief discussion on our

prototype implementation of a fault-resilient version of Cray MPI. The rest of this paper is organized as follows. Section II describes the relevant background information. Section III includes a discussion of some of the new features and optimizations introduced in Cray MPI and Cray SHEMEM. Section IV includes a brief summary of some of the ongoing and future work in Cray MPI and Cray SHMEM.

II. Background

This section provides a brief discussion of the relevant background material.

A. Cray XC Series

The Cray XC series is a distributed memory system capable of sustained multi-petaflops performance. It combines multiple processor technologies, a high performance network and a high performance operating system and programming environments. The Cray XC series uses a novel high-bandwidth, low-diameter network topology called "Dragonfly". The Cray Aries interconnect provides high performance communication in a cost-effective manner. In addition, the Aries network offers a hardware Collective Engine to perform global communication patterns in a highly optimized manner.

The Cray software stack leverages many advanced features of the Cray XC network to provide highly optimized and scalable implementations of MPI, SHMEM, UPC [8], Coarrays [9] and Chapel [10]. These communication libraries are layered on top of the user level Generic Network Interface (uGNI) and/or the Distributed Memory Applications (DMAPP) libraries [11], which perform Cray network-specific operations.

B. MPI Collectives

Collective operations in MPI offer a convenient abstraction layer to implement complex data movement operations. Owing to their ease of use, collective operations are commonly used across a wide range of applications, ranging from turbulence simulations [12] to numerical solvers [13]. In many cases, collective operations involve exchanging data in a coordinated manner across tens of thousands of processes. The current MPI standard also defines non-blocking, or asynchronous collective operations. Parallel applications can utilize non-blocking collectives to achieve computation/communication overlap. It is critical to implement blocking and non-blocking collective operations in MPI in a highly efficient and scalable manner. Cray MPI offers an optimized suite of collective operations. When applicable, Cray MPI utilizes the hardware features offered by the Aries Interconnect to accelerate various collective operations. Section III-A presents a summary of the latest optimizations and features that improve the performance of various important collective operations.

C. MPI-3 RMA

Remote Memory Access (RMA) offers one-sided communication semantics and allows one process to specify all communication parameters, both for the source and target processes. When compared to traditional two-sided point-to-point operations, RMA facilitates the development of scientific applications that have dynamically changing data access patterns. Additionally, RMA operations also remove the need for messages to be explicitly matched between the source and the target processes. Cray MPI offers an optimized RMA implementation by taking advantage of the low level DMAPP [11] interface. This paper presents a summary of the recent enhancements in the RMA interface in Cray MPI (Section III-B).

D. Multi-Threaded MPI Communication

The MPI standard offers various levels of threading support to parallel applications. Parallel applications can request a specific level of support during the initialization, via the MPI_Init_thread(..) operation. It is not uncommon for applications to request MPI_THREAD_MULTIPLE, which implies that multiple threads can concurrently perform MPI operations. However, in order to ensure thread-safety, many MPI software stacks rely on a global lock to serialize the threads. On emerging compute architectures that offer slower scalar speeds, but higher compute density, applications must utilize multiple threads to concurrently perform compute and communication operations. In this context, it is critical to explore design alternatives within the MPI software stack to improve thread concurrency, while also guaranteeing thread-safety. Section III-C summarizes the current status of the new multi-threaded Cray MPI library.

E. Cray SHMEM

Cray SHMEM offers a high performance suite of one-sided communication operations, specifically optimized for the Cray XC series supercomputers. The Cray SHMEM implementation supports remote data transfers via "put" and "get" operations. Some of the other supported operations include broadcast and reduction, barrier synchronization and atomic memory operations. Cray SHMEM software is layered directly on top of the low-level DMAPP API and offers excellent performance for highly parallelized scalable programs.

F. MPI I/O

MPI I/O is a standard interface for MPI applications that do I/O operations. It is part of the MPI standard and a higher level of abstraction than POSIX I/O. POSIX I/O deals primarily with files as a simple sequence of bytes, while MPI I/O can deal with more complex data types. The MPI library converts MPI I/O calls to lower-level I/O calls, but in the process, may also do additional data manipulation based on the data types and optionally perform I/O performance

optimizations for the application. Data-intensive parallel applications spend a significant fraction of their execution time in I/O operations. Hence, it is critical to design solutions to optimize the performance of MPI I/O, and also develop new tools that assist users in understanding the I/O bottlenecks in their applications. Section III-E describes some of the latest work in Cray MPI to improve the I/O performance of scientific applications.

G. Fault Tolerance in MPI

As supercomputing systems continue to scale, they are becoming increasingly vulnerable to hardware failures. Hardware failures often result in the termination of the entire parallel job. Invariably, this leads to inefficient utilization of computing resources. State-of-the-art solutions to addressing this problem include capturing checkpoint images to allow applications to re-start from a previous consistent state. However, there are currently no standardized solutions within MPI to allow application developers to design fault-resilient codes. The MPI community is currently working towards achieving a standardized approach to detect and mitigate system failures. This effort is referred to as the User Level Fault Mitigation (ULFM) [14]. This feature is not yet included in the official MPI specification.

An MPI implementation that offers ULFM support should ensure that any MPI operation that involves a failed process must not block indefinitely. ULFM also mandates that MPI operations that do not involve failed processes must complete normally. ULFM defines extensions to existing MPI interfaces, and new MPI interfaces to detect and manage system failures. For example, a ULFM capable MPI implementation allows a parallel job to re-build MPI objects, such as MPI communicators and windows to exclude dead processes. Parallel applications can utilize the ULFM interface to sustain system failures and continue execution. Section III-F describes a prototype implementation of the ULFM framework in Cray MPICH.

III. Cray MPI and Cray SHMEM: New Designs and Features

This section describes some of the recent features and optimizations in Cray MPI and Cray SHMEM software stacks. The performance data presented in this section were generated using common micro-benchmarks, such as the Intel MPI Benchmark (IMB), and the OSU Micro Benchmarks (OMB). The experiments were performed on Cray XC systems in batch mode.

A. MPI Collectives

MPI_Alltoall and MPI_Alltoallv: Typically, the pair-wise exchange algorithm is used to implement large message All-to-All Personalized Exchange collective operations [15]. The pair-wise exchange algorithm performs the communication operations via point-to-point send/recv operations. Cray MPI also offers an optimized implementation of the MPI_Alltoall and MPI_Alltoallv operations by taking advantage of the low-level DMAPP interface. On Cray XC systems, the DMAPP-based implementation significantly outperforms the basic send/recv-based implementation. However, this feature is not enabled by default, and requires users to explicitly link their executable against the DMAPP library, and enable the corresponding environment variables.

Cray MPI-7.2.0 includes new optimizations to significantly improve the performance of All-to-All Personalized Exchange collective operations. These designs directly leverage the low-level uGNI API and can outperform the default send/recv based implementation of the pairwise exchange algorithm. The new uGNI-based implementation also outperforms the optimized DMAPP-based implementation. Figure 1(a) compares the performance of the uGNI-based implementation with the send/recv implementation. This experiment was performed on a Cray XC system, with 4,096 MPI processes, with 24 MPI processes per compute node. For the purpose of this experiment, we measure the communication bandwidth observed per compute node for various message lengths during the execution of the MPI_Alltoall and MPI_Alltoallv operations. We observe that the new uGNI-based designs can improve the communication bandwidth by about 5X for small message lengths, and continue to perform well for large payload sizes, when compared to the basic point-to-point send/recv implementation. Figure 1(b) compares the performance of the new uGNI-based implementation with the Cray optimized DMAPP-based implementation. The new uGNI-based design outperforms the existing DMAPP-based implementation by 20% for small message lengths, and by up to 15% for larger payloads. Since the uGNI-based solution does not rely on DMAPP, this feature has been enabled by default. Hence, users can automatically benefit from this new optimization. We also note that the use of hugepages is strongly recommended to obtain the best performance. The 64M hugepage module was used for these experiment.

MPI_Allreduce: Cray MPI-7.2.0 offers new optimizations to improve the performance of MPI_Allreduce and MPI_Bcast collective operations. These optimizations rely on implementing intra-node phases of collective operations via buffers dedicated for collective operations in the shared-memory region. Cray MPI relies on the Aries hardware Collective Engine to optimize the Allreduce operation. Figure 1(c) compares the average communication latency for MPI_Allreduce with 119,648 MPI processes on a Cray XC system, with 32 processes per compute node, with and without the new shared-memory optimization. This figure also compares the performance of the DMAPP-based implementation of MPI_Allreduce with and without the shared-memory optimization. We note that Cray MPI-7.2.0 performs about 8% better than Cray MPI-7.0.0 implementation. We attribute this performance improvement to the new shared-memory based optimization. In addition, the

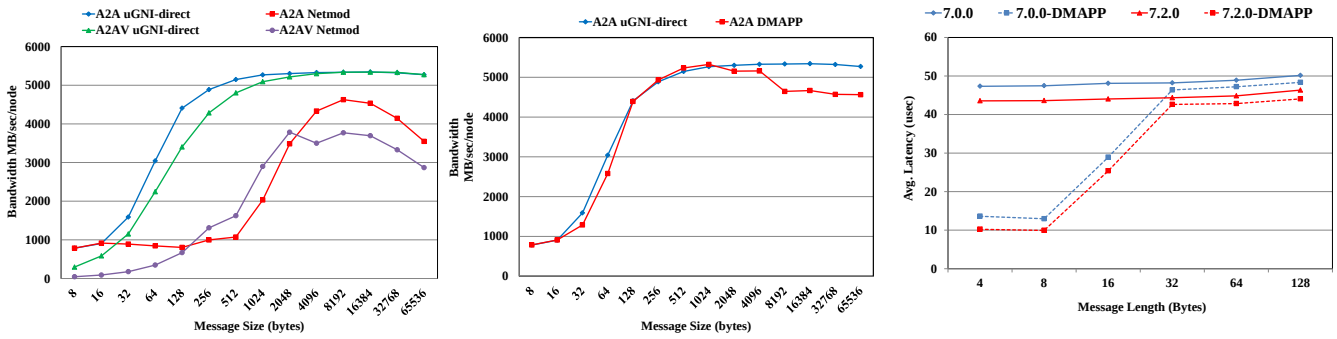

Figure 1. (a) MPI_Alltoall Performance Comparison (b) MPI_Alltoall - uGNI vs DMAPP Comparison (c) MPI_Allreduce Latency Comparison

DMAPP-based implementation of MPI_Allreduce in Cray MPI-7.2.0, together with the new shared-memory optimization performs about 23% better than that of the Cray MPI-7.0.0 implementation. We recommend users to enable the shared-memory based optimization, in conjunction with the DMAPP-based hardware Collective Engine implementation for MPI_Allreduce. The new shared-memory based optimization can be enabled by setting the following flag: MPICH_SHARED_MEM_COLL_OPT. This feature will be enabled by default in upcoming Cray MPI releases.

Non-Blocking collectives: Over the years, Cray MPI has offered support to achieve communication/computation overlap via asynchronous progress mechanisms [16]. Cray MPI-7.2.0 offers improved computation/communication overlap for various non-blocking collectives on XC systems. Cray MPI also offers designs to leverage the Aries Collective Engine support for small message MPI_Iallreduce operations. Figures 2(a) (i) and (ii) demonstrate the latency and overlap characteristics of the improved async-progress optimizations for the MPI_Ialltoall operation, with 8,192 MPI processes on a Cray XC system. For the purpose of these experiments, compute nodes based on the 24-core Intel Ivy-Bridge processors were used. The MPI_Ialltoall benchmark was configured to run with 24 MPI processes per compute node. Cray MPI-7.2.0 offers up to 70% computation/communication overlap for large message Ialltoall operations. In comparison, Cray MPI-7.0.1 offered very little overlap. In addition, the communication latency of the optimized Ialltoall operation is comparable with that of Cray MPI-7.0.0. Similarly, figures 2(b) (i) and (ii) demonstrate the benefits of the new designs with small message MPI_Iallreduce operations. These experiments were also performed on a Cray XC system with 8,192 MPI processes, and 24 MPI processes per compute node. Each compute node was based on the 24-core Ivy-Bridge processor. This design takes advantage of the Aries hardware collective engine to achieve computation/communication overlap. Applications using small message MPI_Iallreduce operations on a Cray XC system can observe up to 80% computation/communication overlap. In addition, the optimized Iallreduce implementation offers lower communication latency, due to the utilization of the hardware collective engine (Section II-A). Optimized asynchronous progress support is not enabled by default. This is because the asynchronous progress design involves spawning an internal helper thread for each MPI process, and utilizing the interrupt-based uGNI transport mechanism. These factors negatively affect the performance of latency sensitive applications. However, this design will improve computation/communication overlap for applications that utilize non-blocking collective operations. Users need to set the following run-time parameters to enable this feature: MPICH_NEMESIS_ASYNC_PROGRESS and MPICH_MAX_THREAD_SAFETY. To enable the MPI_Iallreduce optimization, users need to link against the DMAPP library, and also need to set the following run-time variable: MPICH_USE_DMAPP_COLL.

B. MPI-3 RMA

Cray MPI 7.2.0 offers new designs to improve the performance of some of the one-sided operations by leveraging the hardware atomic operations offered by the Aries network. The hardware atomic operations are particularly well suited for small message RMA operations. In Figure 3(a), we compare the performance of the optimized MPI_Fetch_and_op implementation when compared with the default implementation. The new designs improve the average latency of the MPI_Fetch_and_op operation by more than 80%, when compared with Cray MPI-7.1.3. Users can enable this feature by setting the MPICH_USE_NETWORK_AMO runtime variable.

C. Multi-threaded MPI Communication

The MIC architecture offers more than 50 compute cores per chip. However, the scalar speed of each compute core is much lower when compared to the state-of-the-art Xeon processors. On such architectures, communication libraries that rely on a single thread to service the communication requirements of a parallel application may no longer be able to saturate the high performance communication networks. Parallel applications may need to utilize multi-threaded designs to perform compute and communication operations in parallel across several slower compute cores. In this context, the MPI and SHMEM communication libraries must allow

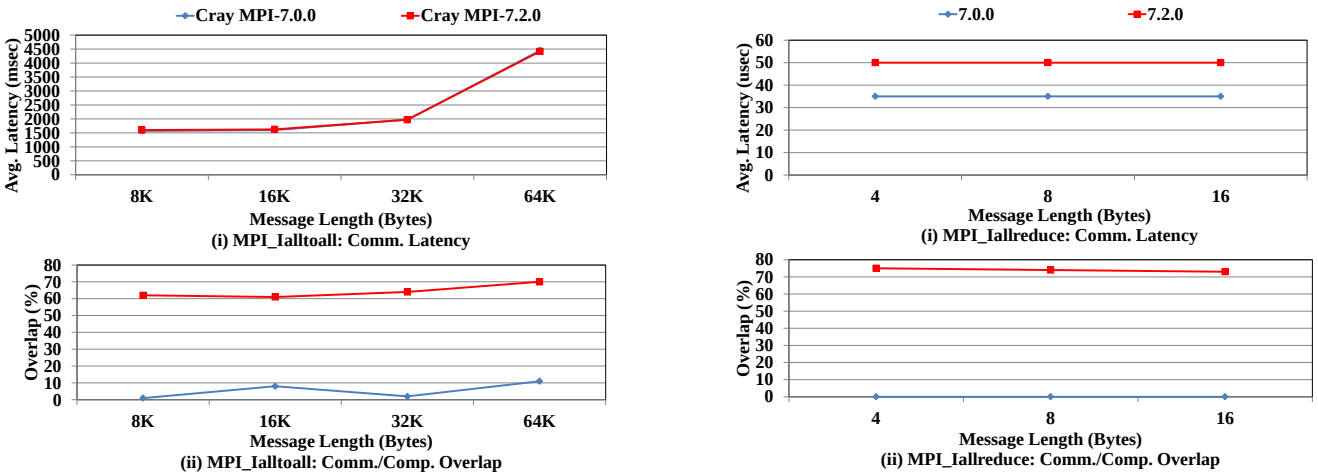

Figure 2. Latency and Overlap Comparison (a) MPI_Ialltoall (b) MPI_Iallreduce

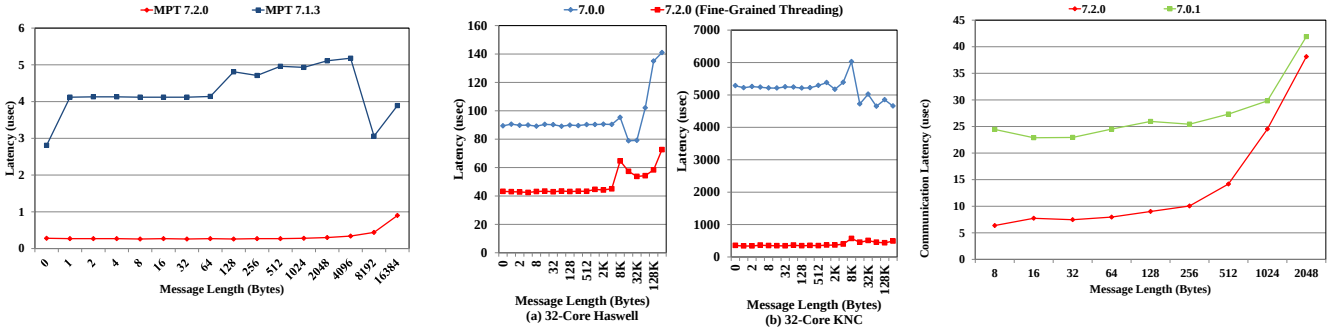

Figure 3. (a) MPI-3 RMA Performance Optimizations, (b) Multi-Threaded Pt2Pt MPI latency, and (c) Cray SHMEM Broadcast Latency

multiple threads to concurrently perform communication operations.

Over the years, Cray MPI has offered a thread-safe MPI software stack by relying on a global lock to guarantee thread-safety. Recently, a new multi-threaded flavor of the Cray MPI software stack was released. This library relies on fine-grained locks to improve concurrency within the MPI library. Figure 3(b) compares the average communication latency of OMB multi-threaded MPI point-to-point benchmark (osu_latency_mt.c). The new library with fine-grained locking significantly outperforms the default Cray MPI library that uses a global lock to ensure thread-safety. We also observe that the new multi-threaded library performs significantly better across two Intel KNC co-processor devices, across the Aries network. Users can link against the new library by using the "-craympich-mt" driver flag, and setting the MPICH_MAX_THREAD_SAFETY run-time parameter to "multiple". The parallel application should also request MPI_THREAD_MULTIPLE during initialization.

D. Cray SHMEM Optimizations

Cray SHMEM 7.2.0 offers new optimizations to improve the performance of the broadcast operation. The new designs are specifically geared towards improving the performance of the broadcast operation with small payload sizes, in the range of 8 bytes to 2KB. In Figure 3(c), we compare the SHMEM broadcast communication latency of the Cray SHEMM 7.2.0, with Cray SHMEM 7.1.0. For small payloads, the new optimizations improve the communication latency by up to 75%.

E. MPI I/O Optimizations

When the performance of the I/O portion of an application is not good enough, it is usually very difficult for the application developer to understand the causes of the poor performance and what might be done to improve it. Cray MPI-7.2.0 offers an MPI I/O performance analysis feature to show I/O patterns on a set of time line plots, giving visibility to and allowing focus on specific, poorly performing parts of the application. Rather than just reporting the total amount of data transferred and the total time spent doing I/O, this feature plots a rich set of I/O pattern statistics as timelines. This provides important insights for possible I/O tuning hints or application modifications. A Cray Apprentice2 feature currently under development will provide even greater visibility of the software stack at any point along the I/O timelines.

Figures 4 (a) and (b) show an example of an improvement

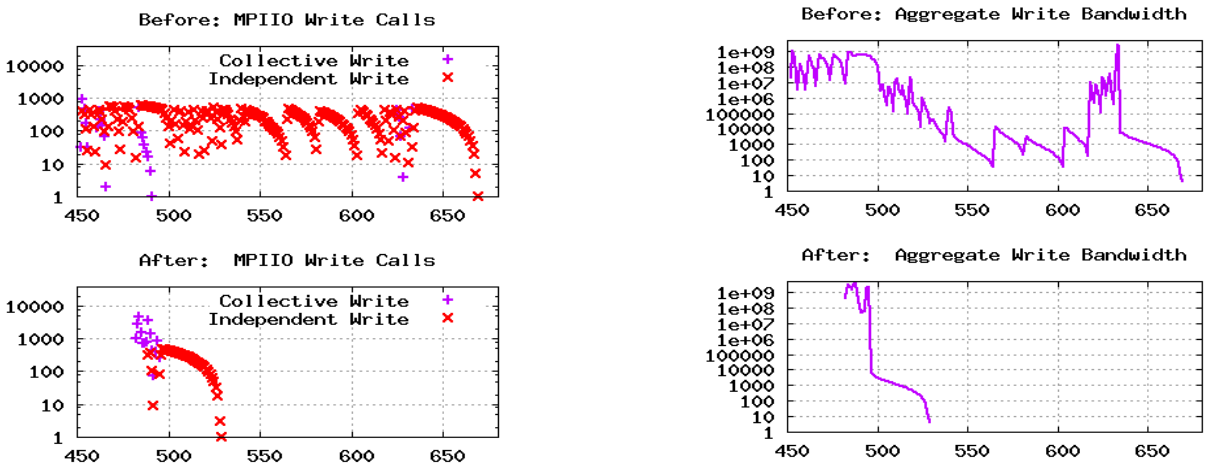

Figure 4. MPI I/O Profile Timeline

made in an application as a result of understanding the file access pattern and then modifying that part of the application for better performance. Figure 4 (a) shows the before and after where the red X's are MPI independent write calls and the blue +'s are MPI collective write calls. The before plot shows that independent calls dominate. Cray MPI cannot do the collective buffering optimization on independent calls. After modifications to the application, most of the MPI I/O calls are collective. The vertical scale is a log scale of the aggregate number of write calls. The horizontal scale is time. The total time for the I/O was reduced from about 200 to about 70. Figure 4(b) shows the before and after for the aggregate write bandwidth. The vertical scale is a log scale of MB/sec and the horizontal scale is time. Most of the low bandwidth in the before case has been replaced with high bandwidth. There is still a tail of independent writes and low bandwidth that could be looked at for more optimization. Users can enable this tool by setting the following environment variable: MPICH_MPIIO_STATS=2.

F. User Level Fault Mitigation (ULFM)

An internal prototype implementation of the Cray MPI with ULFM support is under development. This prototype offers preliminary support for the ULFM framework on Cray XC systems. The current implementation detects node-failures and allows parallel jobs to tolerate such events. Cray MPI also takes advantage of the fault tolerance capabilities offered by the Cray PMI library to dynamically detect failed nodes. The current version of the prototype allows parallel applications to successfully re-build MPI communicators via the new MPI_Comm_shrink, MPI_Comm_revoke() and the MPI_Comm_agree() functions. The prototype implementation of the ULFM framework in Cray MPI is currently not supported in Cray MPI-7.2.0. Future design and development of the ULFM framework in Cray MPI is subject to the ULFM proposal being accepted in the MPI specification.

IV. Summary and Future Work

In this paper, we discussed some of the latest optimizations in Cray MPI and SHMEM software stacks. Several of these optimizations are geared towards improving the performance of scientific applications on modern compute processors and architectures. As discussed in Section III, many new optimizations directly leverage the advanced features offered by the XC network to improve communication performance, communication/computation overlap and I/O performance. Furthermore, a new multi-threaded library has also been released to improve the performance of MPI communication operations in multi-threaded environments. A prototype implementation of a fault tolerant Cray MPI software is also under development. Future releases of Cray MPI and Cray SHMEM software stack will continue to offer new optimizations and features to improve the performance of scientific applications on current and next generation Cray supercomputers.

References

[1] "CRAY: The Supercomputer Company," http://www.cray.com/Home.aspx.
[2] Cray XC Series, http://www.cray.com/Assets/PDF/products/xc/CrayXC30Networking.pdf.
[3] "XEON-PHI Software Developer's Guide," http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-software-developers-guide.pdf.
[4] NVIDIA, http://www.nvidia.com/object/cuda_home_new.html.
[5] MPI Forum, "MPI: A Message Passing Interface (MPI-3)," in *http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf*.
[6] Partitioned Global Address Space, http://www.pgas.org/.
[7] Cray, Inc., "Man Page Collection: Shared Memory Access (SHMEM)," (S-2383-23).
[8] Berkeley UPC, "Unified Parallel C," in *http://upc.lbl.gov/*.
[9] Coarray Fortran, https://gcc.gnu.org/wiki/Coarray.
[10] Cray Chapel, http://chapel.cray.com.
[11] Using the GNI and DAMPP APIs, http://docs.cray.com/books/S-2446-5202/S-2446-5202.pdf.

[12] Parallel 3-D Fast Fourier Transform, http://www.sdsc.edu/us/resources/p3dfft/.
[13] R.D. Falgout, J.E. Jones, and U.M. Yang, " The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners," in *chapter in Numerical Solution of Partial Differential Equations on Parallel Computers, A.M. Bruaset and A. Tveito, eds., Springer-Verlag, 51 (2006), pp. 267-294. UCRL-JRNL-205459.* .
[14] User Level Fault Mitigation, http://fault-tolerance.org/.
[15] J. Bruck, C. T. Ho, S. Kipnis, and D. Weathersby, "Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems," in *Proc. of the 6th ACM Sym. on Par. Alg. and Arch.*, 1994, pp. 298–309.
[16] Howard Pritchard, Duncan Roweth, David Henseler, and Paul Cassella, "Leveraging the Cray Linux Environment Core Specialization Feature to Realize MPI Asynchronous Progress on Cray XE Systems," *Proceeding of the Cray User Group (CUG)*, 2012.