Current State of the Cray MPT Software Stacks on the Cray XC Series Supercomputers

K. Kandalla, P. Mendygral, N. Radcliffe, B. Cernohous, N. Namashivayam, K. McMahon, C. Sadlo and M. Pagel
Cray Inc

{kkandalla, pjm, nradclif, bcernohous, nravi, kmcmahon, csadlo, pagel}@cray.com

***Abstract*—HPC applications heavily rely on Message Passing Interface (MPI) and SHMEM programming models to develop distributed memory parallel applications. This paper describes a set of new features and optimizations that have been introduced in Cray MPT software libraries to optimize the performance of scientific parallel applications on modern Cray XC series supercomputers. For Cray XC systems based on the Intel KNL processor, Cray MPT libraries have been optimized to improve communication performance, memory utilization, while also facilitating better utilization of the MCDRAM technology. Cray MPI continues to improve the performance of hybrid MPI/OpenMP applications that perform communication operations in threaded regions. In addition, Cray MPI is being enhanced to support the MPI Dynamic Process Management Interface and Cray PMI now offers optimized process placement strategies to improve the communication performance of applications on Cray XC systems. Cray SHMEM offers API extensions to create and maintain SHMEM Teams. Finally, this paper describes efforts involved in optimizing real-world applications such as WOMBAT and SNAP on the latest Cray XC supercomputers**

I. Introduction and Motivation

Multi-/Many-core processor architectures, accelerators and high performance networks are enabling the development of highly capable supercomputers. These architectural trends introduce new opportunities and challenges for achieving high performance on modern supercomputers. In this environment, it is imperative to design application and system software stacks in a high performance and scalable manner. Application scientists rely heavily on Message Passing Interface (MPI) and SHMEM programming models to develop scalable distributed memory parallel codes. This paper describes a set of new features and optimizations introduced in the Cray Message Passing Toolkit (MPT) software stacks for modern Cray XC series supercomputers. The Cray MPT product comprises of Cray MPI and Cray SHMEM software libraries that are highly tuned for Cray XC systems. Cray MPI is currently compliant with the MPI-3.1 specification and Cray SHMEM is conformant with the OpenSHMEM-1.3 specification.

The latest Many Integrated Core (MIC) processor, The Intel Xeon Phi. processor, code named Knights Landing (KNL) [1], offers at least 64 compute cores per chip and more than 2 TF double precision floating point performance. However, the performance characteristics of the KNL processor are quite different from those of a contemporary Intel Xeon processor. While the KNL processor offers support for wide vector instructions, the processor cores operate at slower clock frequencies and offer slower scalar processing capabilities. Considering that MPI processing is largely scalar, MPI implementations need to be carefully optimized to improve the communication performance of parallel applications.

KNL also offers an on-package high bandwidth memory technology called Multi-Channel DRAM (MCDRAM) in addition to traditional DRAM. MCDRAM offers high bandwidth (up to 4X more than DDR4) with limited capacity (up to 16GB). MCDRAM can be configured as a direct mapped L3 cache layer or as a distinct NUMA node. Configuring the MCDRAM as an L3 cache layer is a convenient way to port existing applications on to KNL based systems. However, configuring the MCDRAM as a distinct NUMA node and re-designing an application to fully take advantage of the high bandwidth offered by the MCDRAM can improve the performance of memory bandwidth bound applications [2]. The Cray MPT software stack offers new features that can be used by applications to improve the utilization of the MCDRAM and DDR NUMA nodes available on the KNL systems.

II. Key Contributions

This paper describes a range of new optimizations that improve communication latency, bandwidth, and the internal memory utilization of the Cray MPI implementation. Specifically, these optimizations improve off-node MPI point-to-point latency by up to 34% when compared to Cray MPT-7.3.3 and on-node point-to-point bandwidth by up to 2X for a range of message lengths. These optimizations also improve the performance of the Sandia Micro Benchmark message rate benchmark by up to 60% for small messages on Cray XC systems based on KNL processors. A new optimization in the Cray MPI library also improves the latency of small message MPI_Bcast operations by up to 16%. Section IV includes a detailed summary of these new optimizations on Cray XC systems based on KNL processors. While these optimizations specifically target the KNL architecture, many of them are also applicable to the Xeon processors.

Apart from providing support for standard OpenSHMEM-1.3 [3] features, Cray SHMEM continues to support other optimized non-standard features to improve ease of programming and application performance. Existing support for Cray SHMEM teams [4] or flexible PE subsets now includes new team creation and management routines. Optimized team-based collective reduction routines are available with the existing set of other team-based collective operations. The new

team-based collective reduction routines achieve equivalent performance when compared to the OpenSHMEM standard active-set based reduction routines. In addition, collective reduction performance is improved by up to 2X for large messages.

With increasing number of cores per compute node, Cray MPI continues to improve the performance of hybrid MPI/OpenMP applications that perform MPI communication concurrently from multiple threads. [5] discussed the benefits of the "Thread Hot" MPI-3 RMA implementation on Cray XC systems based on Intel Xeon processors. This paper demonstrates the benefits of the optimized MPI-3 RMA implementation on Cray XC systems based on the KNL processors. In addition, Cray MPI offers a new fast and fair locking implementation to improve the performance of point-to-point and collective operations that are performed concurrently from multiple threads. Section IV includes a detailed set of micro-benchmark results to study the performance benefits of optimizing the MPI_THREAD_MULTIPLE threading support in Cray MPI. In addition to the micro-benchmark results, this paper also includes a summary of optimizing and tuning real-world applications such as WOMBAT [6] and SNAP [7]. On Cray XC series supercomputers based on KNL processors, the "Thread-Hot" DMAPP-based MPI-3 RMA implementation in Cray MPT-7.5.3 improves the performance of WOMBAT and SNAP by up to 14% and 22% respectively. Section VII contains a detailed summary of the performance of WOMBAT and SNAP on Cray XC systems based on KNL processors.

Finally, Section VIII describes some of the new and emerging features and optimizations in the Cray MPT software suite. Cray MPI will soon support the MPI Dynamic Process Management feature. The Cray Process Management Interface (PMI) has been extended to expose important system topology information that can be used by user-level applications and libraries to optimize communication performance on Cray XC systems. The latest Cray MPI offers a new lock-ahead optimization for MPI I/O along with exposing internal timers and statistics for I/O performance profiling. Cray MPI also supports a set of user-level statistics that have been integrated with the MPI Tools interface.

The experimental results included in this paper are based on studies performed on Cray XC series systems based on the Aries [8] Interconnect with Intel KNL processors, Intel Broadwell processors and NVIDIA GPUs.

To summarize, the following are the major contributions of this paper:

1) New solutions within Cray MPI and Cray SHMEM to improve the communication performance on Cray XC Series Supercomputers based on the Intel KNL processors.
2) A description of optimizations in Cray MPI to improve the performance of communication operations in Hybrid applications.
3) A summary of enhancements in Cray MPI and Cray SHMEM software stacks to help users best utilize the MCDRAM memory on KNL.
4) A brief overview of new and upcoming features in Cray MPT.

III. Background

This section presents the relevant background information for the various concepts covered in this paper.

A. Support for MPI_THREAD_MULTIPLE in CRAY MPICH

MPI offers three threading modes – MPI_THREAD_SERIALIZED, MPI_THREAD_FUNNELED and MPI_THREAD_MULTIPLE. In the SERIALIZED and FUNNELED modes, only one thread is allowed to issue and progress MPI communication operations. MPI_THREAD_MULTIPLE enables parallel applications to allow multiple threads to perform MPI operations concurrently.

Most MPI implementations (including the default Cray MPI implementation) rely on a single global lock to guarantee thread-safety for the MPI_THREAD_MULTIPLE mode. Each communicating thread must acquire the global lock to perform any MPI operations. Cray MPI also offers an alternate library that relies on fine-grained "per-object" locking mechanisms [9]. In this approach, communicating threads can concurrently enter the MPI library and acquire separate locks that protect different critical sections. This flavor of Cray MPI also uses the global lock to ensure thread-safety for specific components and data structures. Hence, this approach improves the level of concurrency within the MPI library. [5] and [9] discussed the benefits of using the fine-grained per-object locking approach when compared to the single global lock approach. This paper describes a new optimization that is now available in the Cray MPI stack that improves the performance of hybrid applications that perform MPI operations concurrently from multiple threads. This optimization is available in both flavors of the Cray MPI library: the default version that uses the global-lock and the fine-grained "per-object" version.

B. Optimized implementations for MPI-3 RMA in Cray MPI

Cray MPI offers an optimized implementation for MPI-3 RMA operations by leveraging DMAPP [10], the low-level communication library for one-sided data transfers. On Cray systems, [9] demonstrated the benefits of using the DMAPP optimized MPI-3 RMA implementation in Cray MPI when compared to the point-to-point based MPI-3 RMA implementation that is available in the ANL MPICH software. In addition, the MPI-3 RMA implementation in Cray MPI offers "Thread Hot" communication capabilities. This feature is specifically geared towards improving the RMA communication overheads of multi-threaded hybrid MPI applications [5]. Section IV includes a summary of new optimizations and tuning efforts to improve the performance of MPI-3 RMA operations on Cray XC systems based on the Intel KNL processor.

C. KNL MCDRAM and support for Huge page backed memory regions

The KNL processor offers a specialized on package memory called Multi-Channel DRAM (MCDRAM) in addition to the traditional DDR memory. MCDRAM is a high-bandwidth, low capacity memory device that can be configured as a third-level cache, or a distinct NUMA node. When the MCDRAM is configured as a distinct NUMA node in the "flat" mode, applications can benefit if their entire dataset, or the frequently accessed datasets have affinity to the MCDRAM memory. Application developers have four ways of setting memory affinity to the MCDRAM memory: (a) using directives specified by compilers, (b) numactl, or (c) using libraries such as Memkind [11], or (d) by managing memory via OS system calls such as mmap() and mbind(). The Memkind library offers preliminary support for allocating memory regions that are backed by huge pages. This support is currently limited to either 2MB or 1GB page sizes, and may require changes to the OS kernel. Depending on the memory access pattern of a given application, huge pages with 2 MB page size might not be sufficient to achieve high performance communication on a Cray XC system. Allocating all memory regions to be backed by huge pages of 1GB page size is also not a viable solution. Section VI describes a set of new environment variables and API extensions in Cray MPT that enable applications utilize the MCDRAM NUMA node along with huge page support.

D. WOMBAT

WOMBAT is a shock capturing magneto-hydrodynamic (MHD) code used to study a number of astrophysical phenomena including outflows from super-massive black holes [6], the evolution of galactic super-bubbles, and MHD turbulence in environments such as the intra-cluster medium in galaxy clusters. New development is extending the capabilities of the code to cosmological simulations with the addition of a particle-mesh dark matter solver and full multi-grid solver for gravity. This work supports science goals of studying MHD turbulence in galaxy clusters over cosmological scales at very high resolution using a combination of static and adaptive mesh-refinement (SMR and AMR respectively) strategies. WOMBAT development is a collaboration between Cray Inc. Programming Environments and the University of Minnesota - Minnesota Institute for Astrophysics. The performance characteristics of WOMBAT with the Thread Hot MPI-3 RMA capabilities available in Cray MPICH on Cray XC systems based on Intel Xeon processors were studied in [5] and [6]. This paper builds on this effort and evaluates the performance of WOMBAT on Cray XC systems that are based on the Intel KNL processors.

E. SNAP

SNAP serves as a proxy application to model the performance of a modern discrete ordinates neutral particle transport application [7]. It may be considered an update to Sweep3D, intended for hybrid computing architectures. SNAP decomposes a 3D domain with a 2D process grid and each rank managing the full extent of the first dimension. The communication pattern is a wave front with a rank at the corner of the grid producing fluxes that are communicated to the two downstream processes in the Y and Z directions. SNAP has highly optimized computational loops executed in a high level OpenMP parallel region. Non-blocking point-to-point communication is performed by each thread in the parallel region in an effort to overlap it with computation. Since Cray MPT offers a highly optimized "Thread Hot" MPI-3 RMA implementation, Cray has modified SNAP to use MPI-3 RMA operations from within OpenMP parallel regions. Section VII-B studies the performance of the two versions of SNAP on Cray XC systems based on KNL processors.

IV. New Optimizations in Cray MPI for KNL

This section describes a set of optimizations that have been introduced in Cray MPI to improve the performance of various MPI operations.

A. Off-Node MPI Point-to-Point Performance

As part of delivering and optimizing Cray MPI on XC systems based on the KNL processors, the Cray MPT development group has conducted a detailed performance analysis of MPI point-to-point latency, comparing the Intel KNL processors to the Intel XEON processors. It is important to note the Cray MPI library (and code path) is the same on both KNL and XEON processors. When this performance analysis began, MPT 7.3.3 was the state-of-the-art MPT version available on Cray XC systems with KNL processors. Hence, initial results presented in this paper are based on Cray MPT-7.3.3. As discussed in Section II, the KNL processor operates at a slower frequency and offers slower scalar performance. Since the internals of an MPI implementation are largely scalar, it is necessary to minimize the number of instructions executed in the critical code paths inside the MPI library. To this end, the Cray MPT development team explored several optimizations to simplify the polling logic and to reduce the number of function calls in the critical paths of the Cray MPI stack.

Figure 1(a) compares the MPI point-to-point latency observed with Cray MPT-7.3.3 and Cray MPT-7.5.3 for small message off-node ping-pong transfers. Cray MPT 7.5.3 performs about 34% faster for point-to-point off-node transfers when compared to Cray MPT-7.3.3. The Cray MPT development group continues to investigate additional optimizations to improve the communication latency of MPI operations on XC systems based on KNL processors.

B. On-Node MPI Point-to-Point Bandwidth

Apart from optimizing off-node MPI point-to-point performance, Cray MPT also offers enhancements to improve the performance of on-node communication between MPI ranks.

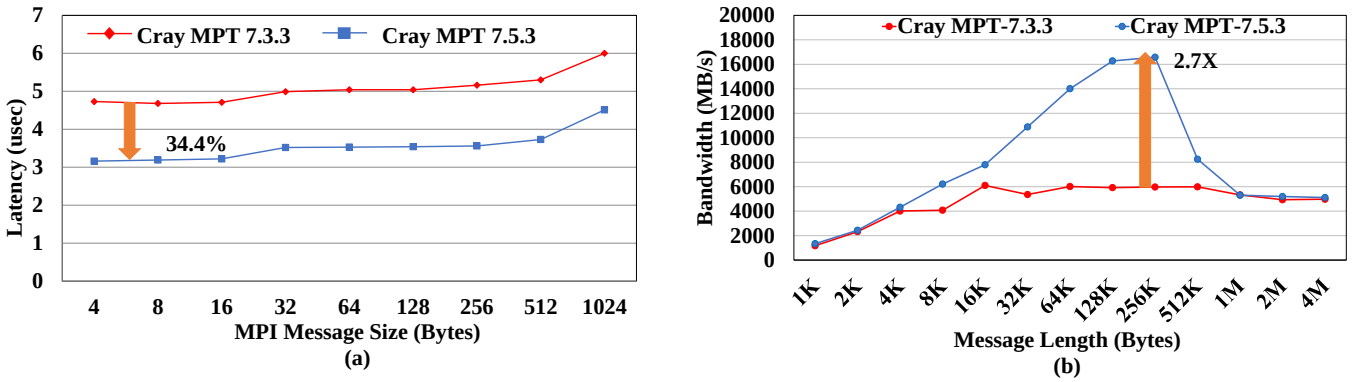

Fig. 1. MPI Point-to-Point Performance Studies: (a) Off-node latency (b) On-node Bandwidth

These enhancements rely on a new `memcpy()` implementation that is specifically tuned for the KNL architecture. Figure 1 (b) compares the performance of Cray MPT versions 7.3.3 and 7.5.3 on the KNL processor. The latest Cray MPT 7.5.3 implementation outperforms the Cray MPT 7.3.3 version by up to 2.7X for on-node communication bandwidth for a range of message sizes that are greater than 1KB. This experiment used with the PrgEnv-gnu module instead of the PrgEnv-cray module. This is because with the PrgEnv-cray module loaded, the Cray compiler software also internally uses the memcpy routines that are optimized for KNL. Hence, using the PrgEnv-gnu module facilitates the comparison of the system default memcpy routines and the Cray optimized memcpy routines on KNL.

C. Off-Node Multi-Pair Latency

Enhancements in Cray MPI also improve the communication performance of MPI point-to-point operations that involve multiple communicating pairs. Figure 2(a) studies the performance of the OSU Multi-pair latency test (osu_multi_lat.c). This benchmark measures the average communication latency between pairs of communicating processes across a range of message lengths. This benchmark was run on an XC system with 2 KNL nodes and 68 processes per node with 32 communicating pairs. The optimized Cray MPT-7.5.3 library outperforms the Cray MPT-7.4.0 library by up to 60% for small messages.

Figure 2(b) studies the performance of the Sandia Micro Benchmark (SMB) with 32 KNL nodes, with 64 MPI processes per KNL (2,048 MPI processes). This experiment used a fixed number of communicating peers (6 and varying the message length from 8B to 2KB. The craype-hugepages8M module was used to perform this experiment. The SMB benchmark reports the communication bandwidth statistics for single-direction, pair-based, pre-post and all-start configurations. For the sake of brevity, this study compares the communication bandwidth reported by the pair-based test between Cray MPT-7.5.3 and Cray MPT-7.4.0. The optimizations discussed in Section IV-A also help improve the performance of the Cray MPT 7.5.3 library with the Sandia Micro Benchmark by as much as 60% for small messages, when compared to the Cray MPT-7.4.0 library.

D. MPI_Bcast Optimizations

Figure 3 studies the performance of the MPI_Bcast collective operation on Cray XC systems based on the KNL processor. Cray MPT 7.5.0 introduces a new design to optimize the performance of the MPI_Bcast operation. This design involves directly issuing data transfer operations via uGNI library calls and by-passing the CH3 channel and the low-level netmod layer within the Cray MPT stack. This approach greatly reduces the number of instructions executed in the critical code-paths, which in turn leads to significantly minimizing the communication overheads involved in the MPI_Bcast operation. This optimization is currently disabled by default. Users must set the MPICH_NETWORK_BUFFER_COLL_OPT environment variable to enable the optimization. The experiment in Figure 3 used 64 KNL nodes, with 64 MPI processes per node (4,096 MPI Processes). For small messages, this optimization improves the communication latency of the MPI_Bcast collective by up to 20%. In Figure 3, the latency of the MPI_Bcast operation in Cray MPT-7.5.0 is considered as the baseline, since this version already includes all KNL related optimizations that have been introduced into the Cray MPT stack since the 7.3.3 version.

E. Threading Optimizations in Cray MPI for off-node point-to-point transfers

As discussed in Section III-A, the Cray MPT suite currently offers two versions of Cray MPI library that offer different levels of support for MPI communication operations in multi-threaded environments. While both libraries support the different threading levels defined by the MPI Specification (MPI_THREAD_MULTIPLE, MPI_THREAD_FUNNELED, MPI_THREAD_SERIALIZED and MPI_THREAD_SINGLE), the default flavor of Cray MPI relies on a single global lock to ensure threads safety. The alternate library relies on using a single global lock when

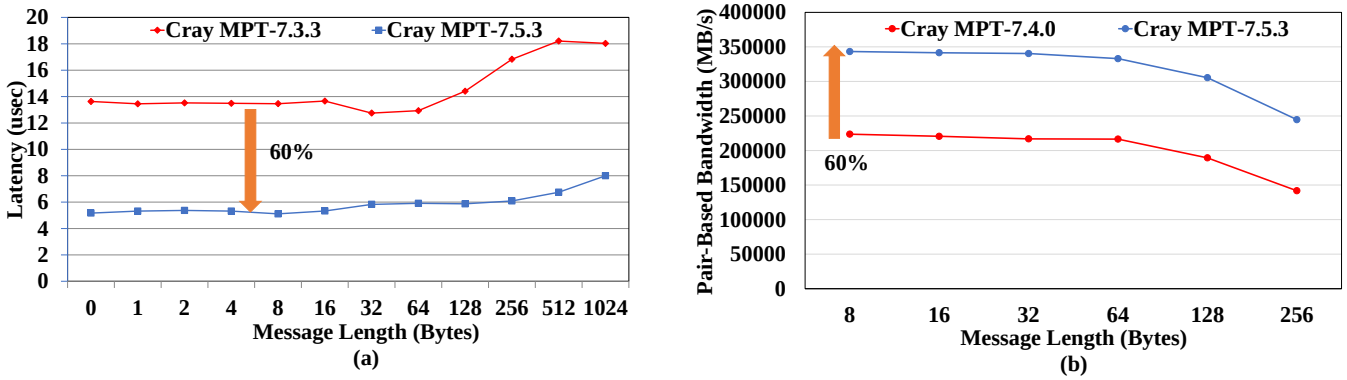

Fig. 2. Multi-Pair Communication Performance Studies: (a) OSU Multilat Benchmark (b) SMB

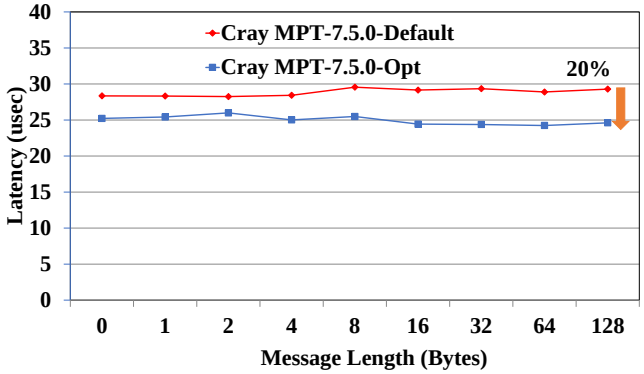

Fig. 3. MPI_Bcast Optimization on KNL

needed, in conjunction with multiple fine-grained per-object locks to improve concurrency inside the MPI library. As documented in the Cray intro_mpi man pages, the alternate library is exposed to the users via a linker flag (-craympich-mt). The fine-grained version of Cray MPI reduces the size of the critical sections and can potentially allow one user-level thread to perform low-level network transactions, while another thread user-level thread operates on higher level MPI constructs, such as the MPI request queue.

Traditionally, both versions of Cray MPI have used the pthread library API to ensure thread safety. While the pthread library offers a standard way of ensuring thread safety in multi-threaded environments, it is not necessarily suitable for high performance communication libraries. For example, the pthread_mutex_lock() API does not guarantee that locks are shared between the blocked threads in a fair manner [12]. Cray MPI 7.5.3 offers an optimization that improves the performance of the global locks in both flavors of Cray MPI library – Global and Per-Object. This optimization implements the recursive global locks by using GCC atomics instead of using the pthread library API. This optimization allows the global lock to be shared between competing threads in a high performance and fair manner. Figure 4 compare the performance of Cray MPI 7.5.0 and Cray MPI 7.5.3 libraries with the osu_latency_th.c benchmark, between 2 KNL nodes on a Cray XC system. MPICH_MAX_THREAD_SAFETY=multiple was set in order to allow multiple user-level threads to perform MPI operations. Each MPI process has 16 communicating threads. This figure also compares the performance of both flavors of the Cray MPI stacks – default (global lock) and the fine-grained per-object lock. For small messages (Figure 4 (a)), the default version Cray MPI-7.5.3 that uses the new global lock implementation performs about 62% better than the default version of Cray MPI-7.5.0 that uses the pthread library to implement the global locks. In addition, the Per-Object (Pobj) version of Cray MPI-7.5.3 performs about 42% better than the Per-Object version of Cray MPI-7.5.0. For larger messages (Figure 4 (b)), the Global-lock version of Cray MPI-7.5.3 performs about 38% better than the Global-lock version of Cray MPI-7.5.0, while the Per-Object version of Cray MPI-7.5.3 is about 13% better than the Per-Object version of Cray MPI-7.5.0. A significant result of this contribution is that the communication performance of multi-threaded point-to-point operations in both the Global and the Per-Object versions of Cray MPI-7.5.3 are now comparable with the osu_latency_mt.c benchmark on Cray XC systems based on KNL processors.

F. MPI3-RMA Performance on KNL

As discussed in Section III-B, Cray MPI offers an optimized MPI-3 RMA implementation by utilizing the low-level DMAPP communication layer [10]. Starting from Cray MPI 7.4.0 version, the RMA implementation is also optimized to offer "Thread-Hot" communication by allowing multiple threads to concurrently issue one-sided operations with minimal resource sharing and lock contention [5]. This implementation was further optimized to achieve very high one-sided message injection rates on Cray XC systems based on the Intel KNL processors. Figures 5 (a) and (b) study the performance characteristics of performing MPI_Get and MPI_Put operations between two MPI ranks on two KNL nodes, each rank performs RMA operations concurrently from 64 threads. MPICH_MAX_THREAD_SAFETY=multiple was

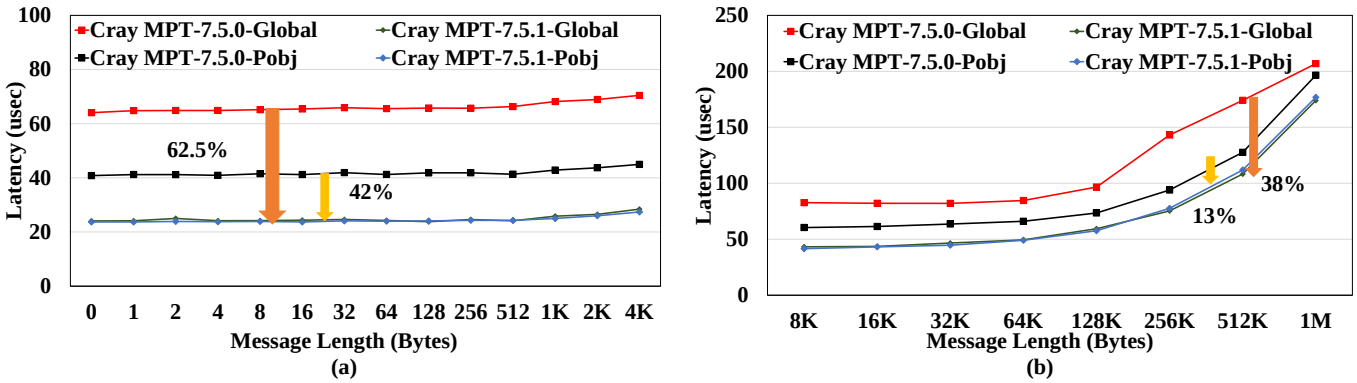

Fig. 4. Cray MPI MPI_THREAD_MULTIPLE communication performance (a) small message (b) large messages

set in order to allow multiple user-level threads to perform MPI operations. In order to enable the Thread-Hot optimization for MPI-3 RMA operations, the executable must be linked against the DMAPP library, and the MPICH_RMA_OVER_DMAPP environment variable must be set to 1. Figure 5 (a) demonstrates that the latest Cray MPI implementation significantly improves the small message communication bandwidth when compared to the DMAPP-based MPI-3 RMA implementation in Cray MPT 7.2.0. This is solely because the Cray MPI-7.5.1 implementation allows each communicating thread to perform the DMAPP-based network operations in an independent manner, whereas Cray MPT 7.2.0 used a single global lock to serialize all network operations that are performed via the low-level DMAPP library. Figure 5 (b) compares the communication bandwidth observed with the same benchmark for large messages. Cray MPT-7.5.0 continues to outperform Cray MPT-7.2.0 for large messages by about 30%.

V. New Optimizations in Cray SHMEM for KNL

This section describes the set of optimizations and new extensions that have been introduced in Cray SHMEM to improve the performance of various SHMEM operations.

A. Off-Node Put Latency

The SHMEM Put operation provides a method to copy data from a contiguous local data object to a remote data object on a specified PE. As per the OpenSHMEM standards, the blocking Put operation returns immediately after the data is copied out of the local data object. In Cray MPT-7.3.3, the default behavior of SHMEM blocking put did not return immediately after the data is copied out. It returned only when the data is copied into the remote data object. The strict completion semantics of blocking put in Cray SHMEM is enforced for thread-safety reasons.

As part of performance improvements in Cray SHMEM, starting from Cray MPT-7.4.0, the default behavior of SHMEM Put operation is enhanced and is now consistent with the blocking Put semantics defined in the OpenSHMEM Specification. Cray SHMEM now allows users to reuse the local data buffer after a shmem_put() operation returns. However, the global visibility of the put operation is not guaranteed until a shmem_fence() or shmem_barrier[_all]() call is completed. A new Cray-specific environment variable (SHMEM_DMAPP_PUT_NBI) is now available. Users can set this flag to 0, if the previous behavior of Cray SHMEM (version 7.3.3) is desired.

Figures 6 (a) and (b) compare the bi-directional write bandwidth with blocking put using Cray MPT-7.3.3 and Cray MPT-7.4.0. Figure 6 (a) compares the performance of Put operations with small message sizes on Cray XC systems based on the Intel Broadwell processor, whereas Figure 6 (b) compares the performance observed on XC systems based on the Intel KNL processor. This experiment uses the SOS [13] bi-directional write bandwidth micro-benchmark [14]. This benchmark uses two PEs on two separate nodes, where both PEs repeatedly perform a stream of `shmem_putmem` operation on a fixed window size before performing memory ordering with `shmem_quiet`, to ensure data delivery from the previous `shmem_putmem` stream. For small data sizes less than 4K, the blocking Put in Cray MPT-7.4.0 performs about 1.8X better than Cray MPT-7.3.3 on BDW and 64% better than Cray MPT-7.3.3 on KNL.

B. Improved API Extensions to Create and Maintain Teams

Many OpenSHMEM routines operate on a subset of PE called *Active set*. The *Active set* is specified using three arguments:

- the lowest PE number of the active set of PEs,
- the log (base 2) of the stride between consecutive PE numbers in the active set, and
- the number of PEs in the *Active set*.

On many HPC applications, using *Active set* for work decomposition is highly insufficient. Cray SHMEM provides a set of Cray-specific extensions for creating and maintaining flexible PE Subsets called *Teams*.

Apart from the existing color- and stride-based Team creation routines, Table I shows the new set of Team creation

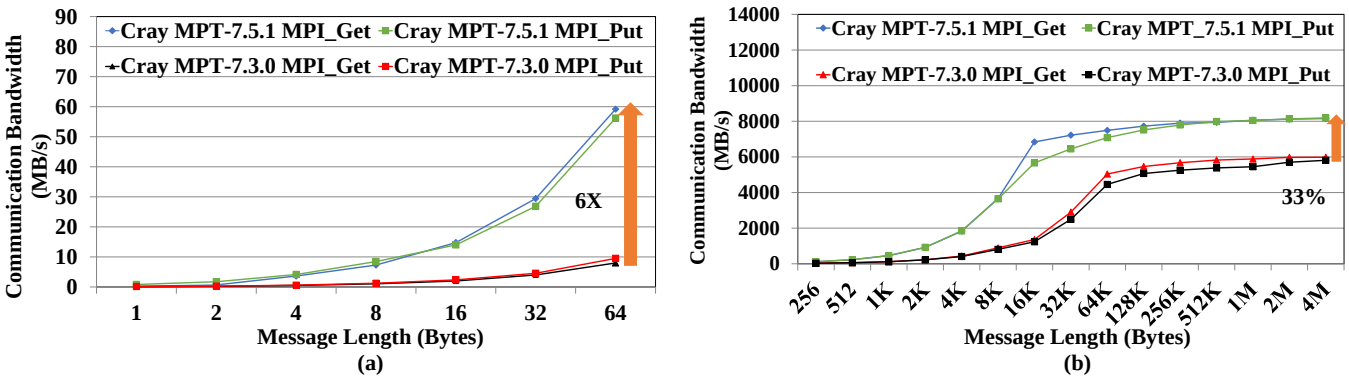

Fig. 5. Multi-Threaded MPI-3 RMA Performance (a) Small Messages (b) Large Messages

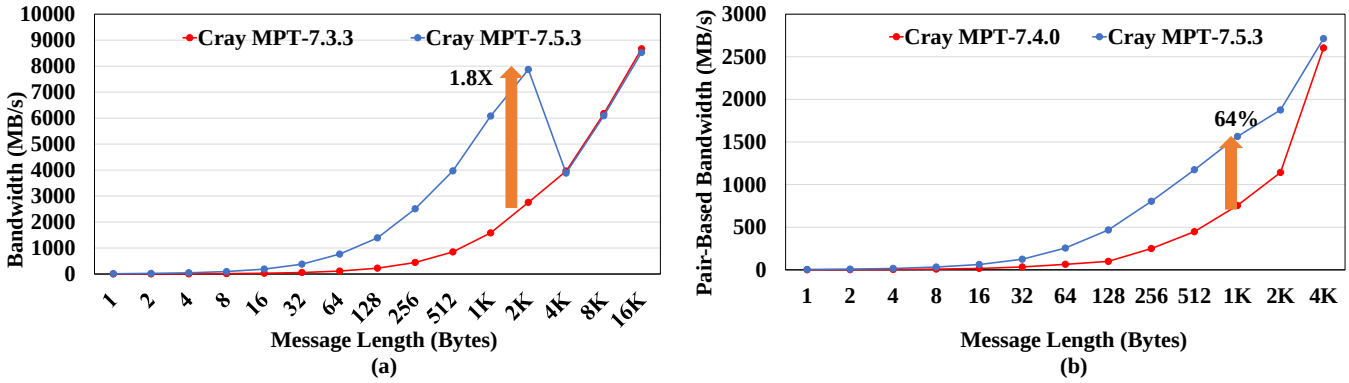

Fig. 6. SHMEM Bidirectional Put Performance Studies: (a) BDW Bandwidth (b) KNL Bandwidth

extensions added to support two and three dimensional Cartesian based Team splits.

2-Dimensional Cartesian splits	`shmemx_team_split_2d ( shmem_team_t parent_team, int xaxis_range, int yaxis_range, shmem_team_t* xaxis_team, shmem_team_t* yaxis_team )`
3-Dimensional Cartesian splits	`shmemx_team_split_3d ( shmem_team_t parent_team, int xaxis_range, int yaxis_range, int zaxis_range, shmem_team_t* xaxis_team, shmem_team_t* yaxis_team, shmem_team_t* zaxis_team )`

TABLE I
TEAM CREATION BASED ON CARTESIAN SPLITS

SHMEM_BARRIER	`shmemx_team_barrier ( shmem_team_t team, long* pSync )`
SHMEM_REDUCTIONS	`shmemx_team_[TYPE]_[OPR]_to_all ( shmem_team_t team, [TYPE]* dest, const [TYPE]* source, int nreduce, [TYPE]* pWrk, long* pSync )`
SHMEM_ALLTOALL	`shmemx_team_alltoall ( void* dest, const void* source, int nelems, shmem_team_t team, long* pSync )`
SHMEM_ALLTOALLV	`shmemx_team_alltoallv ( void* dest, size_t* dest_offsets, size_t* dest_sizes, const void* source, size_t* src_offsets, size_t* src_sizes, shmem_team_t team, long* pSync )`

TABLE II
TEAM BASED COLLECTIVES

In addition to the *Active set* based collective operations, Cray SHMEM also supports *Team-based* collective operations. (Table II).

C. All-reduce collective Performance Improvements

Optimizations in Cray SHMEM improve the performance of OpenSHMEM *Active set* based All-Reduce collective communication for large data sizes greater than 16K. The SHMEM_USE_LARGE_OPT_REDUCE environment variable enables the usage of this optimized collective reduction algorithm and the 16K cutoff size for enabling the usage of this algorithm can be modified using the SHMEM_REDUCE_CUTOFF_SIZE environment variable.

Figure 7 shows the performance improvements observed with the optimized collective reduction algorithm for large data sizes. This experiment used the PGAS All-reduce micro-

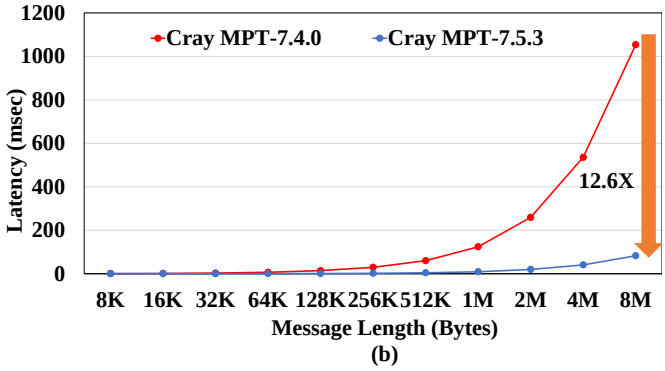

Fig. 7. SHMEM_REDUCE Optimization for large data sizes

benchmark [15]. The optimized collective reduction algorithm performs about 12X better than the default reduction algorithm for very large data sizes. This experiment used 18,000 SHMEM PEs distrubuted across 500 KNL nodes on a Cray XC system.

As mentioned in Section V-B, Cray SHMEM provides support for Team based collective reduction, along with supporting Shared Memory (SMP) based optimizations. SHMEM_TEAM_SMP_REDUCE environment variable is used to enable the SMP-based reduction optimizations. Applications with collective reductions involving the use of two or more PEs per node benefit for the SMP-based optimizations.

Figures 8(a) and (b) shows the usage of SMP-based optimizations on team-based collective reductions using the PGAS micro-benchmarks with 36 PEs per node with 500 KNL nodes (18,000 SHMEM PEs) on a Cray XC system. The SMP optimization improves the communication latency by up to 64% for small messages and by up to 4X for messages of length 16KB.

VI. API Extensions and Environment Variables in Cray MPT to support KNL MCDRAM

This section describes the set of API extensions and new environment variables in Cray MPICH and Cray SHMEM software stacks to enable users better utilize the MCDRAM memory along with hugepages.

A. New Environment Variables in Cray MPI to support KNL MCDRAM

On Cray XC series systems, depending on the communication characteristics of the parallel application, it is beneficial to use huge pages [5]. On KNL based XC systems, if MCDRAM memory configured in the flat mode, application developers now have the flexibility to use hugepage memory regions on both DDR and MCDRAM. As discussed in Section III-C, the programming environment and the WLM offer support to allow users to allocate memory on the MCDRAM, when the MCDRAM is either configured fully or partially in the “flat” mode. However, the current level of support is insufficient to offer memory regions that are bound to the MCDRAM and are also backed by huge pages of a specific page size. Cray MPT software stacks offer solutions to allow users to set specific parameters for various memory allocations. Specifically, users can set the page size, memory policy and memory affinity settings for all memory allocations handled by the MPI_Alloc_mem() and MPI_Win_allocate() operations during the execution of the job. In addition, Cray MPI also offers support to manage the configuration parameters for communication buffers that are allocated by the MPI stack to allow users to better utilize the memory available on the MCDRAM.

MPICH_ALLOC_MEM_AFFINITY specifies the affinity of memory regions allocated by calls to MPI_Alloc_mem() and MPI_Win_allocate(). By default, the memory affinity is handled according to the default affinity settings on a given system (referred to as SYS_DEFAULT). For example, consider a job launched with numactl –membind=1 on a KNL system configured in the Quad/Flat mode. By default, any call to MPI_Alloc_mem() or MPI_Win_allocate() will set the memory affinity to the MCDRAM NUMA node. The MPICH_ALLOC_MEM_AFFINITY variable can be used to override the system default affinity settings. When this variable is set to DDR (or D), memory allocated by MPI_Alloc_mem and MPI_Win_allocate() will be bound to DDR memory. Similarly, if the job was launched with numactl –membind=0 all calls to MPI_Alloc_mem() and MPI_Win_allocate() will, by default, allocate memory with affinity set to DDR. The MPICH_ALLOC_MEM_AFFINITY variable can be set to MCDRAM (or M) to set the affinity of memory regions allocated by MPI_Alloc_mem() and MPI_WIn_allocate() to KNL’s MCDRAM memory.

MPICH_ALLOC_MEM_POLICY specifies the memory affinity policy for memory regions allocated by the MPI library when the user calls MPI_Alloc_mem(), or MPI_Win_allocate(). Suppose MPICH_ALLOC_MEM_AFFINITY is set to MCDRAM. If an MPI_Alloc_mem(), or MPI_Win_Allocate() operation is unable to bind the memory region to MCDRAM, the MPICH_ALLOC_MEM_POLICY variable determines the expected behavior of the program. The Cray MPI library accepts the following policy values: ”Mandatory” (or ”M”), ”Preferred” (or ”P”), or ”Interleave (or ”I”). By default, the memory affinity policy is set to ”Preferred”. This variable has no effect if the MPICH_ALLOC_MEM_AFFINITY variable is not set to ”MCDRAM”. If set to ”Preferred” (or ”P”) and the allocation on MCDRAM is not possible, the MPI library sets the memory affinity to DDR and continues execution. If set to ”Mandatory” (or ”M”), the MPI library fails to allocate memory and displays a meaningful error message. If set to ”Interleave” (or ”I”), the MPI library will try to interleave the individual pages of the allocated memory region across the available MCDRAM NUMA nodes.

The MPICH_ALLOC_MEM_PG_SZ variable allows the user to set a specific page size for MPI_Alloc_mem and MPI_Win_allocate operations. The supported values are

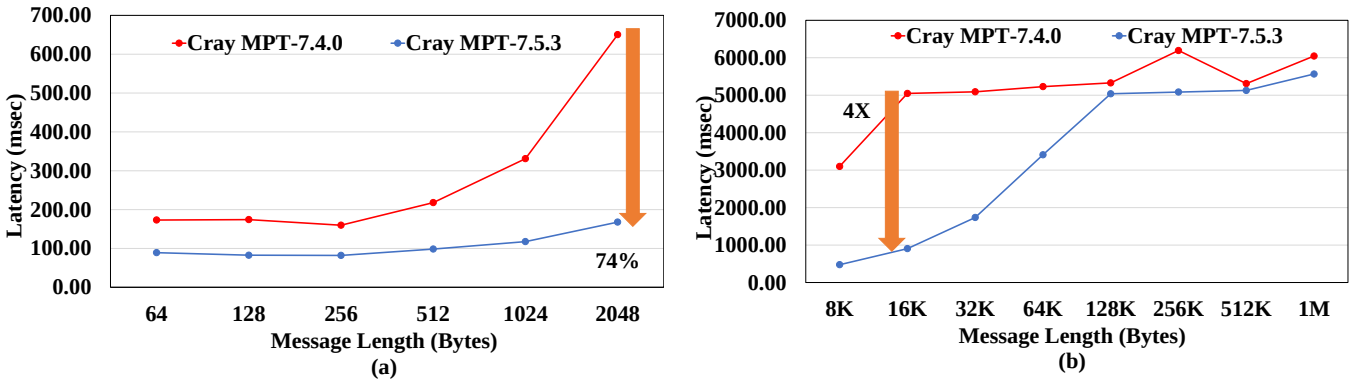

Fig. 8. SHMEM Team-based Reduction Performance Studies with 36 PEs per Node (a) Small Messages (b) Large Messages

4K, 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M and 512M. By default, these operations use the 4K base pages. On Cray XC systems based on the KNL processors, if MPICH_ALLOC_MEM_AFFINITY is set to MCDRAM and MPICH_ALLOC_MEM_PG_SZ is set to X (X>= 2M), the memory region will be backed by huge pages of size X bytes, and the memory affinity will be set to MCDRAM.

Cray MPI offers the MPICH_INTERNAL_MEM_AFFINITY variable to control the affinity of internal memory regions allocated by the MPI library. This variable currently affects the memory affinity of the mail-boxes used for off-node communication, and the shared-memory regions that are used for on-node pt2pt and collective operations. On KNL, this variable allows users to specifically request the internal memory regions used by the MPI library to be bound to either DDR, or the MCDRAM. The default affinity settings will be governed by the system defaults. For example, on a KNL system configured in the Quad/Flat mode, if the job is run with numactl –membind=1, all of MPI's internal memory will be bound to MCDRAM if this variable is not set. This variable can be set to DDR if some of the important memory regions that are used by the MPI library need to be placed on DDR instead.

B. Memory Partitions in Cray SHMEM to support KNL MCDRAM

This section describes the joint efforts of Cray and Intel to define OpenSHMEM runtime changes and extensions to support different kinds of memory for the symmetric heap. At present, symmetric heap is created at a memory location determined by the implementation. The runtime changes in Cray SHMEM allow users to determine the memory location at which the symmetric heaps can be created. These user-determined memory locations are referred to as "Memory Partitions". Each memory partition features a list of traits to define their characteristics – memory kind being one of those featured traits.

We introduce a new environment variable called SHMEM_SYMMETRIC_PARTITION to define the characteristics of each partition by passing one or more traits as inputs. Table III shows the usage of the new SHMEM_SYMMETRIC_PARTITION environmental variable in Cray SHMEM to create memory partitions. At present, we support the following four traits:

SHMEM_SYMMETRIC_PARTITION<ID>= size=<size> [:pgsize=<page_size>][:kind=<mem_kind>][:policy=<mem_policy>]
SHMEM_SYMMETRIC_PARTITION1=size=2G:kind=NORMALMEM
SHMEM_SYMMETRIC_PARTITION2=size=500M:kind=FASTMEM
SHMEM_SYMMETRIC_PARTITION3=size=500M:kind=F:policy=M

TABLE III
ENVIRONMENTAL VARIABLE TO CREATE MEMORY PARTITIONS

- Symmetric heap size - number of bytes to allocate for a symmetric heap and a mandatory input parameter required for each partition,
- Page size - number of bytes to specify the size of the page used by the partition.
- Memory kind - kind of memory if multiple memory kinds are supported by the system. On Intel KNL processors, users can select DDR(NORMALMEM) or on-package MCDRAM(FASTMEM) as input.
- Memory policy - policy to handle scenarios when sufficient memory is unavailable. Users can use MANDATORY or PREFERRED as policy.

Cray SHMEM also provides support for two new extensions `shmem_kind_malloc` and `shmem_kind_align` to create symmetric data objects specific to a memory partition. Users are advised to review the Cray SHMEM man pages for additional information.

VII. OPTIMIZING REAL-WORLD APPLICATIONS ON CRAY XC SYSTEMS WITH KNL PROCESSORS

This section describes a summary of optimizing two real-world applications, WOMBAT and SNAP, on Cray XC systems based on the KNL processors.

A. WOMBAT

Section III-D briefly described the design and implementation of WOMBAT. WOMBAT extensively relies on multiple user-level threads concurrently performing MPI-3 RMA operations to achieve very high efficiency on modern multi-/many-core processor architectures. Cray MPI offers Thread Hot MPI-3 RMA capabilities to multiple user-level threads to concurrently perform RMA communication and synchronization operations. On Cray XC supercomputers based on the Intel Broadwell processors, the Thread Hot MPI-3 RMA optimization in Cray MPICH improves the performance by more than 18% [5]. Figure 9(a) studies the performance of WOMBAT on Cray XC systems based on the KNL processors. Specifically, this study compares the time required to perform a time step in the WOMBAT application which involves both compute as well as communication operations. In addition, this study also elicits the difference in the performance characteristics of WOMBAT by using two versions of Cray MPI: Cray MPT-7.3.1 relies on a global lock to ensure thread-safety for RMA operations, and Cray MPT-7.5.3 offers thread hot RMA communication capabilities. This study was performed on a Cray XC system with 125 KNL nodes, with 4 MPI process per node and 16 threads per MPI rank. Figure 9(a) demonstrates that on XC systems based on the KNL processors, the time required to perform an update is about 14% lesser when using the optimized Cray MPI version that offers Thread Hot MPI-3 RMA communication capabilities.

B. SNAP

The primary communication in SNAP consists of MPI_Isend() and MPI_Recv() called by all threads in a high-level OpenMP region. The communication pattern, described in Section III-E, is a producer-consumer model that is latency sensitive. Traditional point-to-point operations protected with a global lock reduces limits thread scaling because of the added latency from serializing communication. We modified SNAP to use MPI-3 RMA to take advantage of the lower latency and "Thread-Hot" capabilities within Cray MPICH. These modifications consist of changing point-to-point calls into MPI_Put() operations with synchronization handled with a signaling mechanism similar to that used in WOMBAT [6]. A single RMA window and buffer manages both payload and signals in a passive exposure epoch that persists for the full execution time. The RMA buffer is allocated to be large enough to allow independent communication of all energy groups. The main loop over the energy groups is the level at which work is partitioned across threads.

Figure 9(b) compares the solve time for the best performing rank/thread combination for a given number of KNL nodes for the original point-to-point (Pt2Pt) and the new MPI-RMA (RMA) versions of SNAP. The runs are weak scaled with the problem size per node parameters given in Table IV. The RMA version of SNAP performs up to 22% better than the Pt2Pt version with 512 KNL nodes on a Cray XC system. For all RMA runs the optimal configuration was 8 OpenMP threads per rank and 16 ranks per node using 2 Hyperthreads per core. Table V shows the optimal combinations of threads/ranks at a given number of nodes for the Pt2Pt version. The Pt2Pt version is limited to just 4 OpenMP threads and at the largest node counts just a single thread with no Hyperthreads for best performance. In all cases the RMA version enables scaling to higher numbers of threads with better overall performance.

Exp. Type	Threads/Rank	Ranks/Node
Pt2Pt: 2, 16, 64 Nodes	4	32
Pt2Pt: 256, 512 Nodes	1	64
RMA: All Nodes	8	16

TABLE IV
SNAP: CONFIGURATIONS FOR PT2PT AND RMA EXPERIMENTS

Parameter	Value
Nang	48
nx	720
ny	16
nz	16
ng	32

TABLE V
PROBLEM SIZE PER NODE PARAMETERS

VIII. UPCOMING FEATURES IN CRAY MPT

This section profiles two new features that will be available in Cray MPT in the near future.

A. Dynamic Process Management and Improvements in MPI Tag space and message matching

Cray MPT will soon support the Dynamic Process Management interface defined in the MPI Specification. The support for this feature will be rolled out incrementally during Q2 and Q3 of 2017. The Q2 release of Cray MPT will offer support for the MPI_Comm_connect() and MPI_Comm_accept(). Later releases of Cray MPT will support the MPI_Comm_spawn() and MPI_Comm_spawn_multiple(). To use connect/accept in the official Cray MPT release that supports DPM, users will have to set an environment variable MPICH_DPM_SERVER=1 for the server process, and MPICH_DPM_CLIENT="<file path>" for the client process, where "<file path>" is the absolute path to a file created by MPI when launching the server.

B. Improved Network Topology Mapping Interfaces in Cray PMI

Cray PMI now provides a topology-aware, rank-reordering feature that improves communication performance on Cray XC

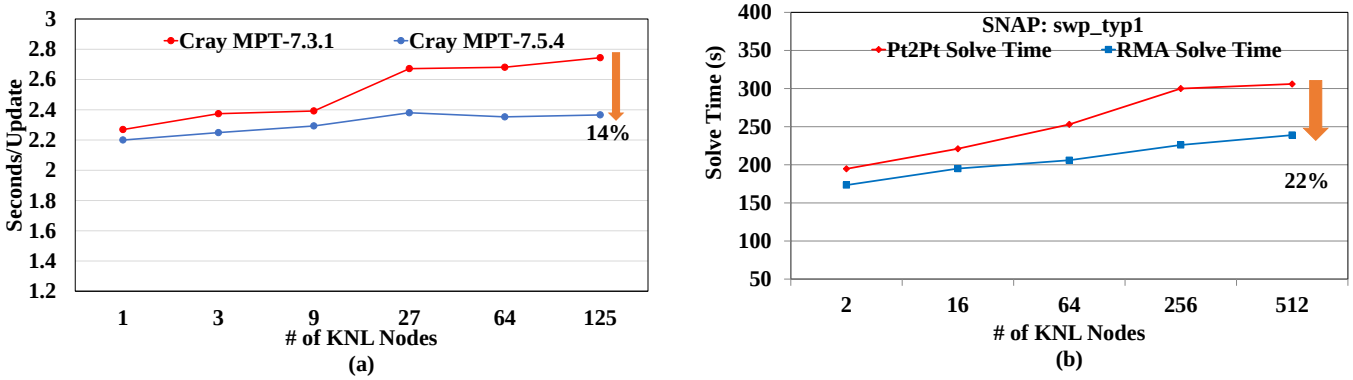

Fig. 9. Application performance on XC systems based on Intel KNL: (a) WOMBAT and (b) SNAP

systems with the Aries Interconnect. In order to use this feature, users must know the communication pattern of their application. The CrayPAT tool can be used on Cray XC systems to obtain this information. Next, users need to set the following environment variable MPICH_RANK_REORDER_METHOD = 4 to override the default rank placement strategy set by the work load manager. In addition, users must set a new environment variable, MPICH_RANK_REORDER_OPTS, with the relevant information for a given job. For example, an application with a 3-dimensional cubic grid, nearest-neighbor communication pattern utilizing 4096 ranks would set: MPICH_RANK_REORDER_OPTS="– –ndims=3 – –dims=16,16,16 – –nearest". During the execution of the job on a Cray XC system, Cray PMI will automatically reorder MPI ranks using a compactness optimization that adapts to the topology of the resources allocated to the user's job to maximize communication locality. For further details, the reader is encouraged to read the "intro_mpi" man-page available a Cray XC system.

Figure 10 studies the performance of the MiniGhost Benchmark on the Trinity XC 40 System. In this experiment, MiniGhost used 8 MPI ranks per node, with each rank having 4 OpenMP threads, across a range of job sizes on the system. The execution time observed with the default process placement for a given job size is compared with the optimized rank placement strategy for the same job size. This study demonstrates that Cray PMI is able to generate highly optimized rank placement strategies for a given job size and this results in an improvement of up to 32% in the overall execution time of the MiniGhost Benchmark.

C. New MPI I/O Optimizations in Cray MPT

Shared-file locking mechanisms available in the Standard Lustre implementations limit scaling of shared-file I/O operations on modern high performance Lustre servers. "Lustre lockahead" is a new Cray enhancement in Lustre to significantly improve write performance for collective and shared-file I/O workloads. Preliminary evaluations show about 200% improvements for small transfer sizes and over 100% improvements for larger transfer sizes, when compared to locking methods available in traditional Lustre implementations. MPI I/O operations in Cray MPT are able to take advantage of this feature to significantly improve shared-file collective I/O performance, achieving more than 80% of file per process performance. [16] describes this new optimization in detail.

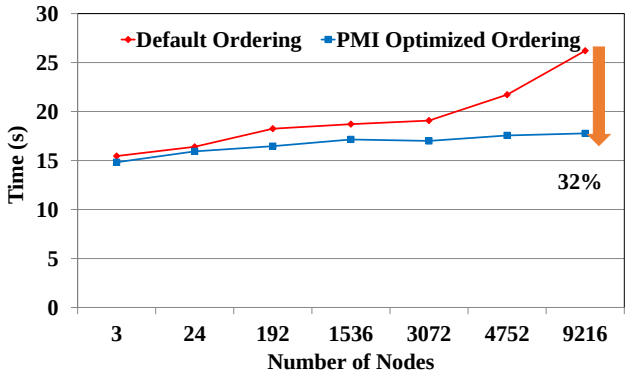

Fig. 10. MiniGhost Execution time on the Trinity XC 40 System

Cray MPT-7.5.3 offers improved support for gathering timing data for MPI I/O operations. MPICH_MPIIO_TIMERS is a new environment variable available in Cray MPT-7.5.3. If this variable is set to 1, timing data for different phases in MPI_IO is collected locally by each MPI process. During MPI_FILE_close the data is consolidated and printed. Some timing data is displayed in seconds, other data is displayed in clock ticks, possibly scaled down. Additional information on the timing data for MPI I/O operations in Cray MPT is available in the intro_mpi man pages.

IX. Related Work

The importance of optimizing the performance of multi-threaded applications is widely recognized. Si [17] et al. have explored the problem of offering a transparent multi-threaded MPI communication library for the benefit of applications that perform MPI operations from a single thread. Amer et al. [18] have explored the problem on optimizing thread

arbitration and scheduling on modern processor architectures. Kumar et al. [19] have also investigated the challenges associated with optimizing multi-threaded MPI communication on IBM BlueGene systems. Balaji et al. [20] proposed the concept of using fine-grained locking mechanisms to improve the performance of multi-threaded applications. This paper discusses the performance characteristics of a new fast and fair thread synchronization mechanism on Cray XC systems. [21] presents the design, implementation and performance characteristics of a SHMEM implementation over the LibFabric [22] interface on Cray XC systems.In [2], the authors discuss the performance characteristics of memory bound applications on KNL processors with MCDRAM configured as a separate NUMA node.

X. Summary and Conclusion

This paper provides a detailed description of a broad range of new features and optimizations to improve the performance of communication micro-benchmarks and real-world scientific applications on the latest Cray XC series supercomputers. The Cray MPT team added new optimizations to improve communication latency, bandwidth, and message rates on XC systems based on KNL processors. Cray MPI also offers several optimizations to improve the performance of applications that use the MPI_THREAD_MULTIPLE threading support. In addition, Cray SHMEM offers enhancements to improve the performance of collective operations that use the *Active set* and *Team* semantics. Nearly all of these enhancements are available in the Cray MPT software stacks. In addition, this paper also demonstrates that the performance of real-world applications such as WOMBAT and SNAP on Cray XC systems based on the KNL processors are improved by up to 22%. Finally, this paper also previews some of the key features and enhancements that will soon be available in Cray MPT.

References

[1] Intel Knights Landing, https://software.intel.com/sites/default/files/managed/e9/b5/Knights-Corner-is-your-path-to-Knights-Landing.pdf.

[2] C. Rosales, J. Cazes, K. Milfeld, A. Gomez-Iglesias, L. Koesterke, L. Huang and J. Vienne, "A Comparative Study of Application Performance and Scalability on the Intel Knights Landing Processor," in *High Performance Computing. ISC High Performance 2016. Lecture Notes in Computer Science, vol 9945. Springer*, 2016.

[3] "OpenSHMEM Application Programming Interface - Version 1.3," http://openshmem.org/site/sites/default/site_files/OpenSHMEM-1.3.pdf.

[4] D. Knaak and N. Namashivayam, *OpenSHMEM and Related Technologies. Experiences, Implementations, and Technologies: Second Workshop, OpenSHMEM 2015*, 2015, ch. Proposing OpenSHMEM Extensions Towards a Future for Hybrid Programming and Heterogeneous Computing, pp. 53–68.

[5] K. Kandalla, D. Knaak, K. McMahon, N. Radcliffe and M. Pagel, "Optimizing Cray MPI and Cray SHMEM Software Stacks for Cray-XC Supercomputers based on Intel KNL Processors," in *Cray User Group (CUG) 2016*, 2016.

[6] P. Mendygral, N. Radcliffe, K. Kandalla, D. Porter, B. Neill, C. Nolting, P. Edmon, J. Donnert and T. Jones, "WOMBAT: A Scalable and High-performance Astrophysical Magnetohydrodynamics Code," *APJS*, vol. 228, p. 23, Feb. 2017.

[7] "SNAP: SN (Discrete Ordinates) Application Proxy," https://github.com/lanl/SNAP.

[8] B. Alverson, E. Froese, L. Kaplan and D. Roweth, "Cray XC Series Network," http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf.

[9] K. Kandalla, D. Knaak, K. McMahon, N. Radcliffe and M. Pagel, "Optimizing Cray MPI and Cray SHMEM for Current and Next Generation Cray-XC Supercomputers," in *Cray User Group (CUG) 2015*, 2015.

[10] Using the GNI and DMAPP APIs, http://docs.cray.com/books/S-2446-5202/S-2446-5202.pdf.

[11] C. Cantalupo, V. Venkatesan, J. Hammond, K. Czurylo, S. Hammond, "User Extensible Heap Manager for Heterogeneous Memory Platforms and Mixed Memory Policies," http://memkind.github.io/memkind/memkind_arch_20150318.pdf.

[12] A. Amer, H. Lu, P. Balaji, and S. Matsuoka, "Characterizing MPI and Hybrid MPI+Threads Applications at Scale: Case Study with BFS," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, May 2015, pp. 1075–1083.

[13] "Sandia OpenSHMEM Implementation," https://github.com/Sandia-OpenSHMEM.

[14] "Sandia OpenSHMEM Performance Benchmarks," https://github.com/Sandia-OpenSHMEM/SOS/tree/master/test/performance/shmem_perf_suite.

[15] "HPCTools PGAS-Microbenchmarks," https://github.com/uhhpctools/pgas-microbench.

[16] M. Moore, P. Farrell and B. Cernohous, "Lustre Lockahead: Early Experience and Performance using Optimized Locking," in *Cray User Group (CUG) 2017*.

[17] M. Si, A. J. Peña, P. Balaji, M. Takagi, and Y. Ishikawa, "MT-MPI: Multithreaded MPI for Many-core Environments," in *Proceedings of the 28th ACM International Conference on Supercomputing*, ser. ICS '14. New York, NY, USA: ACM, 2014, pp. 125–134. [Online]. Available: http://doi.acm.org/10.1145/2597652.2597658

[18] A. Amer, H. Lu, Y. Wei, P. Balaji, and S. Matsuoka, "MPI+Threads: Runtime Contention and Remedies," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP 2015. New York, NY, USA: ACM, 2015, pp. 239–248. [Online]. Available: http://doi.acm.org/10.1145/2688500.2688522

[19] Sameer Kumar, Amith R. Mamidala, Daniel A. Faraj, Brian Smith, Michael Blocksome, Bob Cernohous, Douglas Miller, Jeff Parker, Joseph Ratterman, Philip Heidelberger, Dong Chen, Burkhard Steinmacher-Burrow, "PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2012, pp. 763–773.

[20] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, and R. Thakur, "Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming," *Int. J. High Perform. Comput. Appl.*, vol. 24, no. 1, pp. 49–57, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1177/1094342009360206

[21] K. Seager, S. Choi, J. Dinan, H. Pritchard and S. Sur, "Design and Implementation of OpenSHMEM Using OFI on the Aries Interconnect," in *OpenSHMEM and Related Technologies. Enhancing OpenSHMEM for Hybrid Environments - Third Workshop, OpenSHMEM 2016, Baltimore, MD, USA, August 2-4, 2016, Revised Selected Papers*, 2016, pp. 97–113. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-50995-2_7

[22] LibFabric, http://ofiwg.github.io/libfabric/.