Optimizing Cray MPI and SHMEM Software Stacks for Cray-XC Supercomputers based on Intel KNL Processors

K. Kandalla, P. Mendygral, N. Radcliffe, B. Cernohous, D. Knaak, K. McMahon and M. Pagel
Cray Inc

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

Abstract—**HPC applications commonly use Message Passing Interface (MPI) and SHMEM programming models to achieve high performance in a portable manner. With the advent of the Intel MIC processor technology, hybrid programming models that involve the use of MPI/SHMEM along with threading models (such as OpenMP) are gaining traction. However, most current generation MPI implementations are not poised to offer high performance communication in highly threaded environments. The latest MIC architecture, Intel Knights Landing (KNL), also offers MCDRAM - a new on-package memory technology, along with complex NUMA topologies. This paper describes the current status of the Cray MPI and SHMEM implementations for optimizing application performance on Cray XC supercomputers that rely on KNL processors. A description of the evolution of WOMBAT (a high fidelity astrophysics code) to leverage "Thread-Hot" RMA in Cray MPI is included. Finally, this paper also summarizes new optimizations in the Cray MPI and SHMEM implementations.**

I. Introduction And Motivation

Multi-/Many-core processor architectures and high performance networks are enabling the development of highly capable supercomputers from a hardware perspective. The latest Many Integrated Core (MIC) processor, the Knights Landing (KNL) [1], offers at least 64 compute cores per chip, while offering more than 2 TF double precision floating point performance. However, these trends introduce a new set of challenges and opportunities to improve the performance of parallel applications and system software stacks on modern supercomputers. Over the last two decades, scientific applications have been successfully implemented, tuned and scaled out to tens of thousands of processors using the MPI model. With increasing number of processor cores per compute node, a simple way to effectively utilize the available compute density is to pack as many MPI processes as possible on each compute node. However, on modern architectures, this approach shrinks the fraction of hardware resources that are available for each process and this can potentially limit the overall performance and scalability of parallel scientific applications. In addition, the performance characteristics of a MIC processor is 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 also offer slower scalar processing capabilities. Considering that MPI processing is largely scalar, the MPI-only model alone might not deliver efficient performance on next generation supercomputers.

Recently, hybrid programming models that use MPI in conjunction with threading models (such as OpenMP [2]) are gaining traction. Parallel applications are being redesigned to utilize threading models to leverage multiple compute cores to accelerate computation, while continuing to use MPI [3] to implement inter-process communication and synchronization operations. Hence, hybrid programming models allow parallel applications to utilize the computing power offered by modern multi-/many-core systems, while also allowing MPI processes to own sufficient hardware resources. However, there are several important design considerations that can significantly affect the performance of a hybrid parallel applications. An existing parallel application can be converted to adopt the hybrid programming paradigm by identifying specific blocks of compute that are amenable for threading models. The OpenMP runtime can optimize such blocks of a given parallel application by scheduling work across multiple threads. However, adopting such a "bottom-up" approach might not always ensure the best performance on modern compute architectures. Recent studies have demonstrated that a "top-down" MPI/OpenMP approach can lead to improved performance and scalability when compared to an approach where OpenMP pragmas are applied to specific independent blocks of compute [4].

A Single Program Multiple Data (SPMD) MPI/OpenMP programming approach follows a "top-down" concept and increases the scope of code that are executed by multiple threads and this may result in multiple threads calling MPI operations concurrently. However, most current generation MPI implementations rely on a single global lock to ensure thread safe communication. If multiple user-level threads concurrently perform MPI operations, they all compete for a single global lock within the MPI runtime layer and this poses a significant performance bottleneck on modern multi-/many-core processors. With increasing core counts per node, adopting a highly threaded development approach has the potential to achieve very good computation scaling. However, if a large number of user-level threads contend for a single global lock and the MPI library is fully serialized, it is not feasible to achieve high performance communication for hybrid applications. Because of this limitation, hybrid applications are typically developed to allow a single thread to perform all MPI communication operations. While designing hybrid applications in this manner minimizes the overheads associated with thread synchronization and scheduling, an MPI process relies on a single processor core to progress all communication operations. On emerging processor architectures such as the Intel KNL, this design choice will not be able to offer good communication performance. Hence, current generation MPI libraries and application software layers are not poised to effectively utilize the large number of compute cores on a many-core processor. It is critical to optimize MPI implementations to offer "Thread-Hot" communication, the ability to allow multiple threads to concurrently issue and progress communication operations with minimal synchronization and locking overheads. Furthermore, it is also necessary to re-design hybrid applications in a "top-down" manner to improve computation efficiency and also allow multiple user-level threads to concurrently perform MPI operations.

This paper describes new solutions that are being designed in the Cray MPI software stack to improve the support for multi-threaded MPI communication on modern multi-/many-core processors. In addition, the design, performance and scalability characteristics of WOMBAT, a high performance, scalable astrophysics application is also discussed. WOMBAT is being developed to utilize the MPI/OpenMP model and extensively relies on MPI-3 RMA operations to implement data movement and synchronization operations. It follows a "top-down" MPI/OpenMP model and allows multiple user level threads to concurrently issue MPI-3 RMA operations. This paper studies the performance characteristics of WOMBAT by comparing the default Cray MPI implementation which uses the global lock against the proposed optimized Cray MPI implementation that offers thread-hot capabilities for MPI-3 RMA operations. Preliminary studies demonstrate an improvement of up to 18% in the overall execution time of the WOMBAT application on the Cray XC supercomputer when using the optimized thread-hot Cray MPI implementation. The WOMBAT performance data is based on weak scaling experiments with 34,848 processor cores. These performance benefits are primarily because of the new multi-threading optimizations for MPI-3 RMA operations available in the Cray MPI software stack. The results included in this paper refer to the benefits seen on current generation Cray XC systems with Intel Xeon processors. However, initial studies on experimental KNL hardware have also shown significant promise.

The KNL processor also offers MCDRAM [5], a new memory technology that is designed to improve the performance of memory bandwidth bound applications. The MCDRAM technology can significantly benefit applications that can be modified to fit their entire data sets, or their most commonly used memory regions in the MCDRAM memory. In addition to offering superior memory bandwidth, KNL also offers complex memory hierarchies and NUMA modes. Hence, in order to fully leverage the performance capabilities of this hardware capability, system software stacks (such as MPI and SHMEM [6]) and application software layers must evolve. The Cray SHMEM implementation is being extended to offer new APIs to allow users manage the memory affinity of various memory segments. The Cray MPI implementation is also exposing such controls by extending the existing MPI functions such as MPI_Alloc_mem and MPI_Win_allocate. Specifically, the new features focus on allocating and managing memory regions that are backed by huge-pages on the MCDRAM device. This paper provides an overview of the on-going work in Cray MPI and Cray SHMEM software stacks in this direction. Cray MPI and Cray SHMEM software stacks are available as a part of the Cray Message Passing Toolkit [7] product on Cray Supercomputer.

Finally, this paper also includes a brief summary of a broad range of new features and performance optimizations in Cray MPI and Cray SHMEM software products to improve the performance and memory scalability of parallel applications on Cray XC series supercomputers. To summarize, the following are the major contributions of this paper:

1) New solutions within Cray MPI to improve the multi-threaded communication performance of hybrid parallel applications.
2) A description of the design of WOMBAT, a high performance astrophysics application that relies on optimized multi-threaded MPI-3 RMA implementation in Cray MPI.
3) A summary of proposed enhancements in Cray MPI and Cray SHMEM software stacks to help users best utilize the MCDRAM memory on KNL.

The rest of this paper is organized as follows. Sections II and III describe the major problems being addressed in this paper and present the relevant background material. Sections IV-C and IV-D describe the proposed set of API changes in Cray MPT to facilitate the usage of huge pages on KNL's MCDRAM memory. Sections IV-A and IV-B describe the on-going efforts in Cray MPT to improve communication performance for multi-threaded applications. A detailed description of the design and development methodologies of WOMBAT is presented in Section IV-E. Section V describes the experimental evaluation for the proposed multi-threading optimizations across various communication benchmarks and for the WOMBAT application on a Cray XC supercomputer.

II. Problem Statement

This section describes the major problems being addressed in this paper.

A. Multi-Threaded MPI Communication

A brief description of MPI threading modes and the current state-of-the-art for multi-threaded MPI communication is included in Section III-A. As discussed in Section I, MPI libraries that rely on a single global lock to ensure thread safe communication are not poised to offer the best multi-threaded communication performance. Clearly, new solutions are required within the MPI stack to improve the performance of an emerging class of hybrid applications.

Figures 1 (a) and (b) describe two possible alternatives in designing an optimized MPI stack that can offer improved communication performance for hybrid applications. If an application relies on the MPI_THREAD_MULTIPLE mode, best communication performance can be achieved if the MPI implementation allows multiple threads to drive network transfer operations in a concurrent manner while sharing minimal resources and corresponding locks. This is feasible if the MPI implementation allows each thread to own a different pool of hardware and software resources, which can be accessed with minimal locking and synchronization requirements, as shown in Figure 1 (a). An alternate approach is to follow an "Enqueue-Dequeue" model as described in Figure 1 (b) [8]. Multiple user-level threads enter the MPI library to create and enqueue communication tasks. However, only one thread progresses the communication operations. While this approach can eliminate the need for per-object locks and also reduce the contention for the single global lock, only one processor core can be used by the MPI implementation to progress communication operations. The MPI implementation may be designed to rely on idle OpenMP threads (if any), or on a pool of internal helper threads executing on independent processor cores to accelerate communication progress. However, in this design alternative, the overall performance may depend on the availability of user-level threads, and idle processor cores. If an internal thread pool is utilized to drive communication across multiple cores, the library will require locking mechanisms to ensure thread safety. This paper takes a closer look at the design and implementation issues involved in designing optimized implementations for MPI-3 RMA and the MPI_Alltoall collective operations based on the solution described in Figure 1(a).

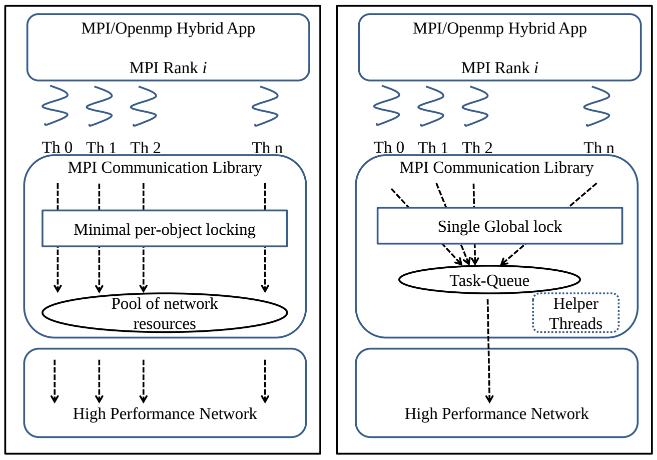


Figure 1(a) Figure 1(b)

Fig. 1. Design Alternatives for Thread-Safe MPI Implementations

B. KNL On-Package Memory (MCDRAM)

Section III-D provides a brief summary of the on-package memory on KNL, along with a high-level description of the existing support that allows users to manage memory on the MCDRAM. Section III-C discusses the importance of managing critical memory regions by leveraging huge pages on Cray supercomputers. On Cray XC systems with KNL, users can continue to load a specific huge-page module in the programming environment and relink the application code to allocate huge-page backed memory on the DDR if they are allocating memory via "malloc()", or "posix_memalign()" calls. However, allocating and managing memory regions that are backed by huge-pages on the MCDRAM are inherently more complex. Loading a specific huge-page module in the programming environment and re-linking the application code does not guarantee huge-page backed memory region if the user is relying on the Memkind library [9] via "hbw_malloc()" to allocate memory on the MCDRAM. As discussed in Section III-C, the Memkind library offers basic support for allocating memory regions that are backed by huge-pages. Depending on the memory access pattern of a given application, the level of support offered by the Memkind library might not be sufficient to offer the best communication performance on the Cray XC. Clearly, solutions within programming models that allow users to allocate specific memory regions to be backed by a range of huge-page sizes and memory kinds is necessary. Programming models must also handle scenarios where the KNL nodes are configured to expose only a part of the MCDRAM as a separate NUMA node. Under such configurations, sufficient memory might not be available to offer specific huge-page support for the entire application. Programming models must either transparently fall back to allocating memory on the DDR, or report a fatal error in such cases. In addition, a SHMEM implementation must also expose an appropriate level of support to allocate the symmetric memory regions on the MCDRAM with a consistent huge-page size across all KNL nodes. This paper describes our ongoing efforts to address these requirements.

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 [10]. 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. On current generation Cray XC systems, the fine-grained per-object locking approach delivers significant performance improvements for multi-threaded MPI point-to-point communication benchmarks. A prototype implementation of the Cray MPI library that minimizes the need for the global locks is under development. *Brief-Global* is another approach to locking that involves using a single global-lock carefully with smaller critical sections. While this design alternative can greatly simplify the implementation details and code maintenance the MPI implementation is still largely serialized.

B. *Optimized implementations for MPI_Alltoall(v) and MPI-3 RMA in Cray MPI*

Cray MPI offers highly optimized implementations for heavy data moving collectives such as MPI_Alltoall and MPI_Alltoallv. These implementations have been demonstrated to significantly outperform the pair-wise exchange algorithm that is available in the ANL MPICH implementation on the Cray XE and XC systems [10]. Cray MPI uses a novel communication and synchronization algorithm by directly relying on the low-level uGNI [11] library to optimize the communication performance of MPI_Alltoall and MPI_Alltoallv operations. This also allows the implementation to bypass the netmod layer to implement the data transfer operations. However, the Alltoall(v) implementations require all the participating processes to exchange meta-data information about the Alltoall operations, which includes the memory addresses of the communication buffers and the corresponding network registration handles with each other prior to implementing the uGNI communication operations. This meta-data exchange is done at the start of each MPI_Alltoall(v) operation and is handled by the netmod layer in the Cray MPI stack. This implementation is ideally suited for use cases where only one thread is performing the Alltoall operation because it relies on a single global lock to ensure thread safety.

Cray MPI also offers an optimized implementation for MPI-3 RMA operations leveraging DMAPP [11], the low-level communication library for one-sided data transfers. The DMAPP optimized RMA implementation has also been demonstrated to significantly outperform the basic MPI-3 RMA implementation that is available in the ANL MPICH software stack. This implementation also relies on a single global lock to ensure thread safety and is not poised to offer high performance communication for hybrid applications.

Section IV includes a summary of new optimizations to improve the performance of MPI-3 RMA operations and MPI_Alltoall in multi-threaded environments. Cray MPI 7.3.2 already offers the advanced support for “thread-hot” MPI-3 RMA operations and allows multiple user-level threads to concurrently perform RMA operations in a high performance and scalable manner.

C. *Significance of using huge pages on Cray Supercomputers*

On the Cray XC series systems, it is important to manage frequently accessed memory regions by backing them with huge pages. Memory regions that are backed by huge pages significantly reduce the pressure on the Aries TLB and this can improve network communication performance. The Cray programming environment allows users to load a specific huge page module and re-link their codes to automatically gain the benefits of using huge page backed memory regions. Furthermore, Cray MPI automatically manages several internal communication buffers by using huge pages to improve communication performance. In addition, Cray MPI also allows users to allocate memory backed by huge pages of specific sizes via the MPI_Alloc_mem() interface. Users can call MPI_Alloc_mem() and set the “MPICH_ALLOC_MEM_HUGE_PAGES” environment variable to instruct the MPI implementation to allocate memory regions that are backed by huge pages. The MPICH_ALLOC_MEM_HUGE_PAGES variable is not set by default, and a call to MPI_Alloc_mem() defaults to using malloc(). If the variable is set, the Cray MPI implementation allocates memory that is backed by huge pages and the default page size is 2MB. The page size can be configured by setting “MPICH_ALLOC_MEM_HUGEPG_SZ”. The reader is advised to refer to the Cray MPI man pages for additional details about these environment variables.

D. *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 [9], or (d) by managing memory via OS system calls such as mmap() and mbind(). The Memkind library also 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.

E. 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 [12], the evolution of galactic super-bubbles, and MHD turbulence in environments such as the intra-cluster medium in galaxy clusters. Additional components available in the code can be enabled to incorporate the effects of static and time-dependent gravity, acceleration and aging of cosmic rays, and radiative cooling. 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 is being done to support 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 is being developed through a collaboration between Cray Inc. Programming Environments and the University of Minnesota - Minnesota Institute for Astrophysics.

IV. Design

A. Optimized MPI_THREAD_MULTIPLE support for MPI-3 RMA in Cray MPI

Thread-hot MPI-3 RMA is designed both for high bandwidth, high message rate multi-threaded communication, as well as contention-free communication and message completion. This last point is important, as it allows the user to flush outstanding messages on one thread while other threads continue to make uninterrupted progress driving further communication.

The optimized Cray MPI allocates network resources to threads in a dynamic manner. In the proposed implementation, the pool of network resources available to each rank is static and resources cannot be shared between ranks. Since network resources are not statically assigned to each thread, the design can scale up to any number of threads per rank. If the number of threads per rank that are simultaneously driving communication exceeds the number of network resources available to threads on that rank, then there will be contention for network resources; otherwise, the design is contention-free. This is true even when various threads are simultaneously making both communication and message completion calls, such as MPI_Win_flush. Epoch synchronization calls, such as MPI_Win_complete and MPI_Win_fence, are thread-safe, but not intended to be used in a thread-hot manner; multiple threads calling, say, MPI_Win_start and MPI_Win_complete will open and close multiple RMA epochs, rather than using multiple threads to speed up a single epoch synchronization.

All RMA communication calls, including request-based versions, are thread-hot. So are MPI_Win_flush and related operations. Thread-hot communication is possible using any type of epoch synchronization, but passive synchronization is likely the best fit for thread-hot communication; MPI_Win_flush can be used to complete messages in a multi-threaded code region without forcing bulk-synchronization of all outstanding messages, and MPI_Win_flush_all can complete all outstanding messages without blocking other threads from communicating or completing messages. MPI_Win_flush(win, rank) will complete all outstanding messages on ”win” that targeted ”rank”, and that were initiated before the call to MPI_Win_flush. It is the responsibility of the user to ensure proper ordering of calls by multiple threads to RMA communication functions and MPI_Win_flush, whether through locks, atomic counters, or some other means.

B. Optimized MPI_THREAD_MULTIPLE support for MPI_Alltoall in Cray MPI

As discussed in Section III-B, Cray MPI software offers a high performance, scalable implementation for MPI_Alltoall and MPI_Alltoallv collective for single threaded use cases. The design objectives of thread-hot MPI_Alltoall collective operation is to allow multiple user-level threads to concurrently issue and progress the collective with minimal locking and synchronization requirements. This approach can potentially improve the performance of a “top-down” MPI/OpenMP hybrid application that performs the MPI_Alltoall collective in a multi-threaded code region.

The proposed thread-hot optimization for MPI_Alltoall is based on concepts discussed in Figure 1 (a) and is consistent with the proposed thread-hot designs for MPI-3 RMA in Section IV-A. The design allocates a pool of network resources for each rank. Each communicating thread is dynamically assigned a set of network resources to allow multiple threads to simultaneously drive communication operations without the need for additional locking and synchronization mechanisms. The number of hardware resources that are allocated per MPI process can be configured via an environmental variable. If the number of communicating threads exceeds the number of available network resources, the design relies on a fine-grained lock to ensure thread-safety. Lock contention is observed only in the scenario where the number of communicating threads exceeds the available set of hardware resources.

The proposed thread-hot implementation significantly reduces the need for locks during the data movement operations. However, the optimized implementation requires the use of locks around the meta-data exchange phase (Section III-B) of the Alltoall implementation because it is handled by the Cray MPI netmod layer. The overheads associated with this phase of the Alltoall implementation depends on the number of user level threads concurrently posting the Alltoall operation, along with the level of load imbalance across the threads. The meta-data exchange can be implemented in a thread-hot manner if the underlying netmod layer in Cray MPI can also be re-designed to allow multiple threads to make concurrent progress with minimal locking requirements. These optimizations are currently under investigation and will be available in the future. Additionally, the meta-data exchange phase can be eliminated if the MPI standard offers the use of persistent collectives [13]. This can allow multiple user-level threads to start, progress and complete their Alltoall operations with minimal locking requirements to offer high performance communication in highly multi-threaded environments. Section V-B compares the performance of the proposed thread-hot implementation when compared to the default uGNI Alltoall implementation in Cray MPI for varying number of MPI processes and threads per MPI process.

C. API Extensions and Environment Variables in Cray MPI to support KNL MCDRAM

As discussed in Sections III-D and II, 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. Upcoming Cray MPI releases will offer this feature.

Cray MPI will allow users to request a huge page backed memory region that is bound to the MCDRAM with a specific page size. This feature will be exposed via the MPI_Alloc_mem() and the MPI_Win_allocate() operations. A brief description of the environment variables and their usage is described in Figure 2. Section III-C described the MPICH_ALLOC_MEM_HUGE_PAGES and MPICH_ALLOC_MEM_HUGEPG_SZ environment variables. The official Cray MPI release for KNL will slightly modify these environment variables to better suit the various memory kinds and policies available on KNL. Hence, the MPICH_ALLOC_MEM_HUGE_PAGES and MPICH_ALLOC_MEM_HUGEPG_SZ variables will soon be deprecated. A new environment variable MPICH_ALLOC_MEM_AFFINITY allows users to set the affinity of a requested memory region to either "DDR" or "MCDRAM". The MPICH_ALLOC_MEM_POLICY allows users to specify a memory allocation policy on the KNL architecture – "Preferred", "Mandatory" or "Interleave". Finally, the MPICH_ALLOC_MEM_PG_SZ variable allows users to specify the page size for the MPI_Alloc_mem() and MPI_Win_Allocate() operations. Users are allowed to request a page size in the following range: 4K, 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M and 512M. This feature will initially be exposed to users via environment variables. Future releases of Cray MPI will offer info key support for finer grain control of memory placement, page sizes and memory policy.

By default, MPICH_ALLOC_MEM_PG_SZ is set to 4KB. If the user requests memory affinity to be set to MCDRAM (by setting MPICH_ALLOC_MEM_AFFINITY to MCDRAM), if sufficient MCDRAM is available, the MPI library will return a memory region that is backed by 4KB pages and bound to the MCDRAM device. If the user adjusts the page size to 128MB, Cray MPI implementation will return a memory region that is backed by 128MB huge pages, on the MCDRAM device. On current KNL systems, MPICH_ALLOC_MEM_POLICY defaults to "Preferred". This allows a parallel job to run to completion even if sufficient memory is not available on the MCDRAM device. This setting can be overridden by setting the memory policy to "Mandatory" to trigger a fatal error if sufficient memory is not available on the MCDRAM device to service a specific memory allocation request.

On KNL nodes that are configured to use SNC2 and SNC4 NUMA modes, the Cray MPI library will allocate memory on the closest memory node, regardless of the MPICH_ALLOC_MEM_AFFINITY value. For example, if the KNL is configured in the SNC4 mode , MPICH_ALLOC_MEM_PG_SZ is set to 128MB and the memory affinity is set to MCDRAM. If an MPI process is scheduled on core id 55, the MPI library will allocate memory on the MCDRAM device that is closest to core id 55 (which is NUMA NODE 8). In addition, this memory region will be backed by 128MB huge pages.

If MPICH_ALLOC_MEM_PG_SZ is not set, the Cray MPI library will return memory regions that are backed by 4KB base pages either on DDR or MCDRAM, depending on the MPICH_ALLOC_MEM_AFFINITY value. In this scenario, the MPICH_ALLOC_MEM_POLICY variable has no effect.

MPICH_ALLOC_MEM_AFFINITY = MCDRAM or DDR
If set to DDR, MPI_Alloc_mem() allocates memory
with affinity set to DDR.
If set to MCDRAM, memory affinity is set to MCDRAM
(Default: DDR)

Use MPICH_ALLOC_MEM_PG_SZ to adjust the page sizes
(Default: 4KB (base pages). Allowed values: 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M, 512M)

MPICH_ALLOC_MEM_POLICY = M/P/I
M: Mandatory (Fatal error if MCDRAM allocation fails)
P: Preferred (Fall back to using DDR if MCDRAM unavailable)
I : Interleaved (memory affinity is set to interleave across MCDRAM NUMA nodes)
(Default: P)

If base pages are used, ALLOC_MEM_POLICY defaults to PREFERRED.

Fig. 2. Env. Variables offered by Cray MPI to manage Huge page backed memory regions on KNL

D. API Extensions in Cray SHMEM to support KNL MCDRAM

This section describes the joint efforts of Cray and Intel to define an OpenSHMEM API that will provide a portable way of allocating and using heterogeneous memory kinds on future processor architectures. Specifically, this section concerns the development of new API that targets the on-package MCDRAM memory on the Intel KNL processor. Figure 3 offers a high-level description of the proposed API. This capability allows the user to utilize different page sizes across the DDR and the MCDRAM memory kinds on the KNL. The proposed API extension offers new environment variables to allow users to specify the memory characteristics of one or more symmetric memory partitions. Users can specify a maximum of SHMEM_MAX_PARTITIONS, and the partition numbers may be in the range 1 through SHMEM_MAX_PARTITION_ID. These environment variables can be used to specify the size, the kind of memory, memory policy and the desired page size. The memory kind may be either "D[efault]" or "F[astmem]". This API allows the programming model to offer additional memory kinds depending on the memory hierarchy of future processor architectures. "page_size" refers to the size of the pages for the specific memory partition and can be one of the following strings: 4K, 2M and 1G. As discussed in Section IV-C, "policy" allows the programming model to handle scenarios where sufficient memory is not available to handle a specific user request.

```
SMA SYMMETRIC PARTITION1=size=<size>
                           [:kind=<kind>]
                           [:policy=<policy>]
                           [:pgsize=<page size>]
SMA SYMMETRIC PARTITION2=size=<size>
                           [:kind=<kind>]
                           [:policy=<policy>]
                           [:pgsize=<page size>]
<size> is the size in bytes for the memory partition.

<kind> can be one of these strings:
  D[efault]
  F[astmem]

<policy> can be one of these strings:
  M[anditory]
  P[referred]
  I[nterleaved]

void *shmem kind malloc(size t size, int partition id);
void *shmem kind align(size t alignment, size t size, int partition id);
```

Fig. 3. Proposed SHMEM API Extensions for managing memory on KNL

E. Optimizing WOMBAT for Multi-/Many-Core Architecture

Section III-E provided the relevant background information for WOMBAT. To achieve the scientific goals for WOMBAT several challenges must be addressed, and this presented opportunities to design the code with considerations for future architectures, such as KNL. The simulations for studying MHD turbulence in galaxy clusters will require excellent scaling well beyond 10^5 cores. Because of this, algorithms were chosen to avoid the need for global or long distance communication. Among the time consuming solvers, each is expressed as nearest neighbor or user-defined local sub-volume communication only. With communication traffic localized, the most significant remaining issue is load balancing. Cosmological SMR/AMR simulations naturally lead to load imbalance among MPI ranks when the simulated volume is decomposed in space.

As dark matter clumps and plasma follows, MPI ranks that enclose such volumes will have to perform more work due to the increased density of dark matter particles and refinement of the grid. Some of this work must be explicitly moved from overloaded rank to other ranks with less to do. One simple way to reduce the frequency of load balancing is to decompose the global grid over fewer MPI ranks and keep the core count constant by utilizing OpenMP threads. Increasing the sub-volume of the data grid that each rank operates upon reduces the effect of additional load relative to the base workload. For simulations with some level of periodicity in space, such as cosmology, larger MPI rank sub-volumes also increases the likelihood of ranks retaining workloads similar to other ranks over time. If threading is properly implemented, it should then be trivial to load balance within an MPI rank by utilizing one of the load balancing OpenMP loop schedules without needing to explicitly move any data. WOMBAT was designed for this reason to push OpenMP thread scaling to the limit of a single MPI rank per node and utilize all cores on a node with threads.

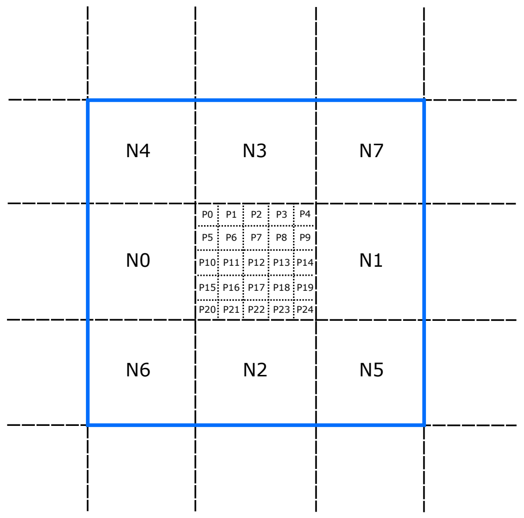


Fig. 4. WOMBAT: Mult-Level Domain Decomposition in 2D

MPI programming often exposes more parallelism than traditional compute loop-based OpenMP techniques produce. The top-down model for MPI increases the likelihood of concurrency between processes down the stack of an application relative to the use of bottom-up OpenMP threading around specific time-consuming loops. Unless enough of an application has been multi-threaded and that threading eliminates data motion or synchronization bottlenecks from MPI, it is far from clear that a hybrid version will perform better than pure MPI. This issue was central to the design of WOMBAT and how OpenMP was used. The development methodologies and performance characteristics of WOMBAT are also applicable to other MPI/OpenMP hybrid applications on current and emerging compute architectures.

The approach for threading in WOMBAT was to move the OpenMP parallel region to the top of the call stack with all threads executing nearly all of the code. Data structures were designed such that a minimum of coordination between threads was necessary. The most important of these are structures that decompose an MPI rank's portion of the world grid by another level. Figure 4 describes the domain decomposition approach in WOMBAT with a two-dimensional example. The world grid is subdivided into domains for each MPI rank. Here a single domain is focused in the center with neighboring rank domains labeled N0-N7 around it. Each domain is further decomposed into sub-grids. In this example a 5x5 decomposition is used inside each rank domain, labeled P0-P24 for the rank in focus. These sub-rank grids each have their own boundaries that can be exchanged within a rank with explicit copies or between ranks with MPI. The size of these sub-rank grids is tunable, and typically the number of them (e.g., hundreds) far exceeds the number of cores on a node (e.g., tens). With each sub-rank grid being a completely self-contained portion of the world grid, a significant amount of parallel work is exposed that is easily threaded and load balanced.

The result of the sub-rank decomposition done in WOMBAT is a significant increase in the number of MPI messages and a reduction of the typical message length relative to having a single large grid within a rank. This pushes the communication characteristics closer to message rate sensitivity. Since MIC architectures offer slower serial performance, seeking out an MPI feature easily optimized for message rate is necessary. Additionally, the increase in the number of messages may result in exhaustion of the number of available MPI message tags. MPI-RMA is ideally suited to addressing both of these issues. Similar to the compute parallelism exposed by the sub-rank decomposition, an equal amount of communication parallelism is exposed. The thread-hot MPI-RMA feature available in Cray MPI allows threads to process this communication work concurrently, which results in significant performance improvement. Section V-C provides a detailed experimental evaluation of WOMBAT on Cray XC supercomputers with the default Cray MPICH implementation, and the optimized thread-hot implementation.

V. Experimental Evaluation

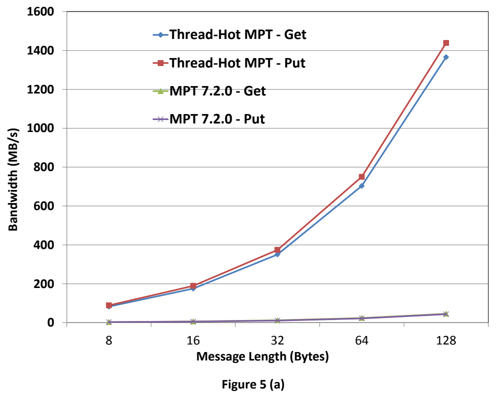
A. MPI_THREAD_MULTIPLE support for MPI-3 RMA in Cray MPI

The MPI-3 RMA communication benchmarks that are available in the OSU Micro Benchmark suite (OMB) have been modified to allow multiple user-level threads to issue RMA operations concurrently. The benchmarks have been designed to use as many as 32 OpenMP threads per MPI process, each thread performing RMA operations (such as MPI_Get, MPI_Put) concurrently with specific source/destination processes. The benchmarks report communication bandwidth for MPI_Put and MPI_Get operations, with message sizes ranging from 8 Bytes up to 1 MegaByte. Cray MPI with MPICH_RMA_OVER_DMAPP enabled is considered as the baseline for these experiments. In the benchmark, a single rank with 32 threads targets three separate off-node ranks, but never itself. For each message size, there is an inner and an outer loop. The outer loop is an OpenMP "for" loop, with a "guided" schedule and 4096 iterations. A new target rank is selected at the beginning of each iteration of the outer loop, and the target remains consistent throughout the inner loop. The inner loop has 128 iterations, and each iteration makes a single call to MPI_Put or MPI_Get, depending on the benchmark. After the inner loop completes, MPI_Win_flush is used to complete all messages issued during the inner loop.

Figure 5 (a) and (b) demonstrate the improvements observed for the communication bandwidth for MPI_Put and MPI_Get across various message lengths. This experiment was performed with two compute nodes on a Cray XC system with Intel Broadwell processors, with one MPI process per node. Each MPI process has 32 communicating threads. Cray MPT 7.2.0 is used as the baseline for this comparison, along with data from the prototype implementation (Thread-Hot MPT) described in Section IV-A. These studies demonstrate that the bandwidth improvements are more pronounced for smaller message sizes, but there are noticeable improvements even for larger message sizes. We still expect to see some improvements for small message bandwidth in thread hot MPI RMA due to further code optimizations, especially through the use of DMAPP's "non-blocking implicit", or NBI, functions, which chain multiple messages into a single network transaction. The thread hot RMA designs described in Section IV-A are already available in the Cray MPT 7.3.0 software package. Future versions of Cray MPT will include additional optimizations that rely on DMAPP's NBI functions. Moreover, the thread hot optimization for RMA will be available as a part of the default Cray MPI implementation. This allows users to directly benefit from this optimization without having to link against an alternate library, or by enabling any other environment variables apart from MPICH_RMA_OVER_DMAPP.

B. MPI_THREAD_MULTIPLE support for MPI_Alltoall in Cray MPI

Section IV-B describes the on-going efforts in Cray MPI to offer thread hot communication capabilities in Cray MPI software. The proposed design is intended for use cases that follow a "top-down" MPI/OpenMP Hybrid application development approach that involve multiple user level threads concurrently performing MPI_Alltoall in a concurrent manner on different communicators. The communication benchmark used for this study is based on the Alltoall collective benchmark available in the OSU MPI Benchmark Suite (OMB) [14]. This benchmark extends the osu_alltoall benchmark to allow multiple user-level threads to perform MPI_Alltoall in a concurrent manner.



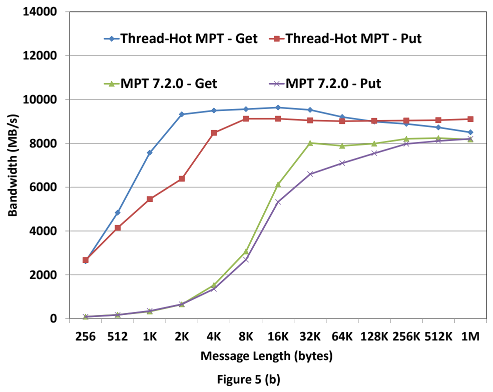


Fig. 5. MPI_Put and MPI_Get Bandwidth comparison: Default Cray MPI (7.2.0) and Prototype Cray MPI implementation with thread hot RMA (a) Small message lengths (b) Large Message lengths

Since collective operations do not accept tag values, each thread performs the Alltoall collective on a different MPI communicator handle. The benchmark calls MPI_Comm_dup to duplicate MPI_COMM_WORLD to create a distinct copy of MPI_COMM_WORLD for each communicating thread. Since each process now relies on multiple threads to perform the Alltoall operation, the payload size handled by each thread is a function of the original payload size and the number of communicating threads per process. Figures 6 compare the execution time of the multi-threaded Alltoall benchmark for 64 nodes (256 MPI processes) and 128 nodes (512 MPI processes), with 4 and 8 threads per MPI process, across a range of message lengths. These experiments were performed on a Cray XC system with Intel Broadwell processors. The "-d" and "-S" aprun options were used to set the affinity of the MPI processes and their corresponding threads. Figures 6 demonstrate that the proposed thread hot MPI_Alltoall implementation improves the communication latency reported by the multi-threaded communication benchmark, when compared to the serial version of the benchmark with one thread executing the Alltoall operation. In addition, these experiments also demonstrate that the thread hot implementation outperforms the Global-Lock and the Per-Object-Lock Cray MPI implementations. Also of importance is the fact that the MPI communication latency often degrades when multiple threads are concurrently performing MPI operations. This is observed in Figure 6(c), where the global-lock and the per-object-lock implementations have higher communication latency than the "default" implementation. This is to be expected because of lock contention within the MPI library as the number of user level threads that perform MPI operations increase. However, the proposed thread hot implementation performs about 10% better than the default implementation. Additional optimizations are being performed and this feature will be released as a part of the Cray MPI implementation in the near future.

C. *Scaling results with WOMBAT on Cray XC systems*

This section describes the experimental analysis for WOMBAT on Cray XC systems. The performance and scalability characteristics of WOMBAT is compared between the default Cray MPI implementation that offers thread safety by relying on a global lock and the proposed thread hot Cray MPI implementation described in Section IV-A.

Figure 7 shows an example of WOMBAT strong thread scaling on a dual socket XC40 Intel Haswell node for a fixed grid using a single MPI rank, with varying number of threads. Runs with turbo enabled and disabled (p-state uncapped or capped at stock frequency) are shown along with a theoretical linear speed-up curve for reference. This experiment demonstrates that the WOMBAT performance is very close to the theoretical speed-up curve when the threads having affinity to the same NUMA domain. With increasing number of threads, the number of cross-NUMA transfers increase, which leads to a slight degradation in the speed-up when compared to the theoretical baseline.

Figure 8 (a) shows the weak scaling of WOMBAT on an XC system with Intel Broadwell processors. In this experiment, a 3-D test problem is considered with the number of MPI processes ranging from 1 to 968, with each rank having 36 threads. When only a single MPI rank is used there are no calls to MPI, and boundary exchanges between sub-rank grid is all done with explicit copies. The number of MPI_PUT and MPI_GET calls increases as more ranks are used until it saturates at 27 ranks (864 cores). Curves for MPT 7.3.1 and the optimized version of Cray MPT that offers the thread hot MPI-RMA feature are shown. MPT 7.3.1 does not have the thread hot MPI-RMA feature, and all calls are protected by a single big lock. The proposed Cray MPI implementation (future release) introduces the thread hot feature, and the time to complete an update in the test simulation drops by over 18%. This is direct result of threads being able to process MPI communication concurrently significantly reducing MPI-related overhead costs. Cray MPT 7.3.2 already offers the thread hot MPI-RMA capabilities. Future releases of Cray

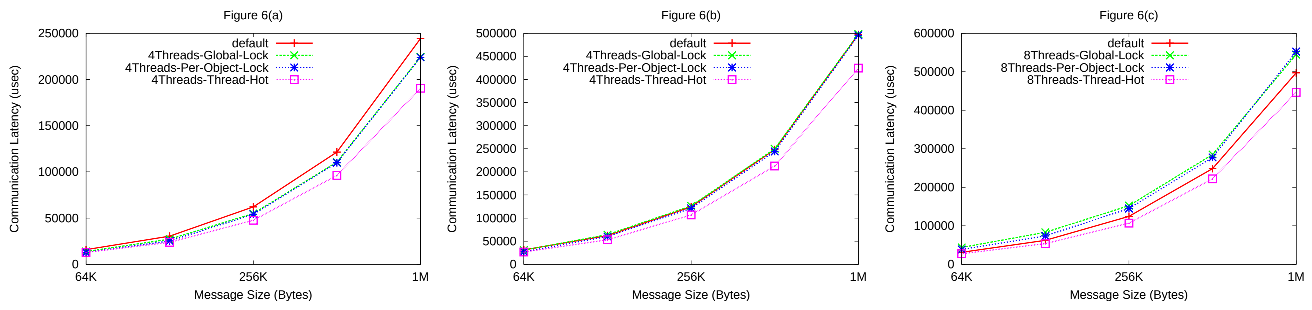


Fig. 6. Execution time of MPI/OpenMP MPI_Alltoall Benchmark: (a) 64 Nodes, 256 MPI processes, 4 threads per process; (b) 128 Nodes, 512 MPI Processes, 4 threads per process; (c) 128 Nodes, 512 MPI Processes, 8 threads per process

MPT (Q4, 2016) will offer additional optimizations that allow multiple threads to perform RMA synchronization operations in a concurrent manner.

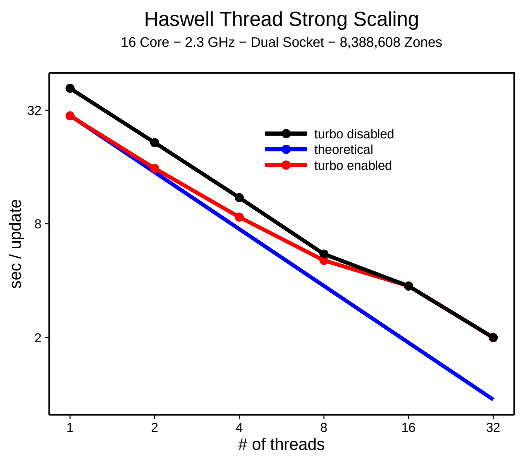


Fig. 7. WOMBAT thread strong scaling on a 32 core XC Haswell node for a three dimensional test problem run on with a single MPI rank

Figure 8(b) shows how WOMBAT performs for a three dimensional fixed grid calculation at 34,848 cores going from MPI only (single thread per rank) to wide OpenMP (36 threads per rank, single rank per node). WOMBAT spends very little time in message synchronization, typically around 1%, and generally achieves very good overlap of computation and communication. With this property on a fixed grid calculation, there is no attribute of WOMBAT or the algorithms in use that prefers threads over ranks. Therefore ideal performance would show ranks with 36 threads performing just the same as ranks with just a single thread. Figure 8(b) shows that performance varies by less than 8% between these two extremes, which demonstrates that the thread implementation of both WOMBAT and Cray MPICH MPI-RMA is very efficient. Additional optimizations to improve concurrency in RMA synchronization operations are currently in progress in Cray MPI. This optimization is expected to further reduce the performance difference between high and low thread counts shown in Figure 8(b). Ultimately the simulations required for the science goals of WOMBAT will prefer threads to ranks for load balancing reasons. Those runs are expected to show superior performance of very high thread counts over low thread counts.

Figure 8(c) demonstrates the relative performance improvements observed on the Intel KNL processors on Cray XC systems. This figure shows the performance improvements observed with WOMBAT by using the proposed thread hot MPI-RMA optimizations in Cray MPI, when compared to the default implementation of Cray MPI that relies on a single global lock to ensure thread safety. WOMBAT runs with fewer than 68 cores show no performance benefits because there are no MPI-RMA operations and all data movement operations are implemented via memory copies. However, as WOMBAT is scaled to use larger number of cores on the KNL processor, the performance offered by the proposed thread hot MPI-RMA optimization outperforms the default Cray MPI implementation by about 40% on Cray XC systems based on the Intel KNL processor. This experiment demonstrates the significance of developing hybrid parallel applications and the MPI implementations in a highly optimized manner.

VI. Related Work

The importance of optimizing the performance of multi-threaded applications is widely recognized. Si [15] 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. [16]

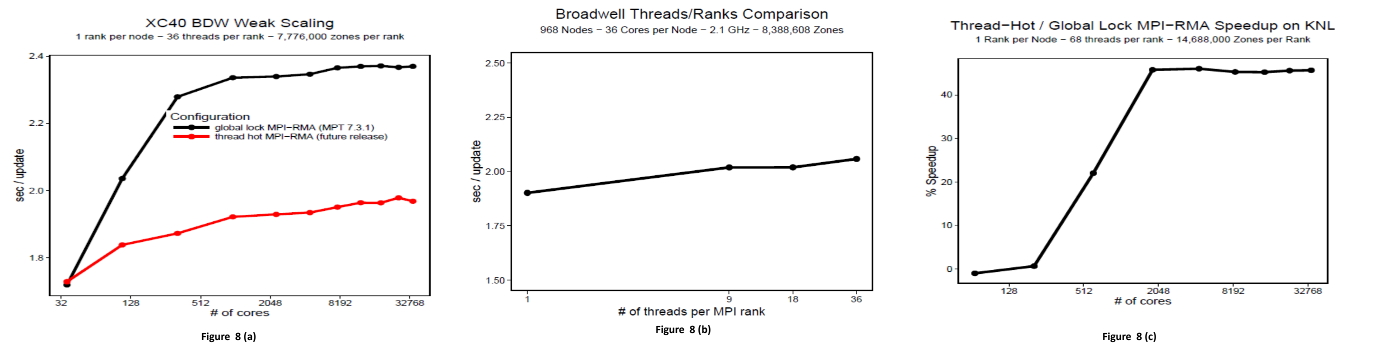


Fig. 8. WOMBAT Application Weak Scaling analysis on XC: (a) Weak Scaling on Cray XC with Intel Broadwell processors, (b) Thread/Rank Comparison (c) WOMBAT Application Weak Scaling on KNL Processors (Relative Data)

have explored the problem on optimizing thread arbitration and scheduling on modern processor architectures. Kumar et al. [17] have also investigated the challenges associated with optimizing multi-threaded MPI communication on IBM Blue-Gene systems. Balaji et al. [18] proposed the concept of using fine-grained locking mechanisms to improve the performance of multi-threaded applications. The default MPICH library from ANL relies on a single global lock to ensure thread-safety. Many MPICH derivatives use the same approach to offer thread safe communication for specific network interfaces. However, the Cray MPI library also offers preliminary support for fine-grained multi-threaded communication progress by utilizing per-object locking mechanisms [10]. Conceptually, per-object locking mechanisms utilize smaller locks around specific critical objects. While these locks guarantee correct multi-threaded access to global MPI objects, they also allow multiple threads to make concurrent progress within the MPI library because the size of the critical section is much smaller when compared to the single global lock implementation. Cray MPI has extended this design to offer fine-grained multi-threaded performance on the Cray XC series supercomputer systems. However, in the current version of the per-object Cray MPI implementation, the netmod layer is still being guarded by a global lock. This solution is currently available as a non-default MPI library. Users can specify a driver flag to link against the optimized multi-threaded MPI library and are encouraged to read the Cray MPT man pages to learn more about this feature. Preliminary experiments have shown promising results for multi-threaded point-to-point communication operations. A prototype implementation of the Cray MPI library that minimizes the need for using a global lock within the uGNI netmod layer is under development. This library will be available for experimentation on the Cray XC series systems in the near future.

VII. Summary and Conclusion

In this paper, we explored the significance of designing MPI/OpenMP Hybrid parallel applications in a "top-down" manner to improve the overall computation and communication efficiency of WOMBAT, a high performance astrophysics application. We used this case study to motivate the need for MPI implementations to offer efficient communication performance for highly multi-threaded applications where multiple user level threads may call MPI operations concurrently. Novel solutions were proposed to implement MPI-3 RMA and MPI_Alltoall operations in a "Thread-Hot" manner to allow multiple threads to independently drive network communication operations with minimal locking and resource sharing. Detailed experimental evaluations were performed on Cray XC supercomputers to understand the performance characteristics of the proposed thread-hot MPI implementation with various multi-threaded communication benchmarks with different processor counts, across various message lengths. An in-depth study was also included to demonstrate that the overall execution time of the WOMBAT application can be improved by more than 18% with more than 34,000 cores on Cray XC supercomputers based on Intel Haswell and Broadwell processors. Experiments on early KNL hardware have demonstrated very promising results with scaling WOMBAT with the proposed thread-hot Cray MPI implementation. In addition, this paper also proposed new API changes in Cray MPI and Cray SHMEM to facilitate the usage of huge page backed memory regions on KNL processors. These features are expected to greatly improve the performance of applications running on Cray systems based on Intel KNL processors. A version of thread-hot MPI-3 RMA implementation is already available in Cray MPT 7.3 version. Future releases of Cray MPT will offer additional optimizations for MPI-3 RMA, a thread-hot MPI_Alltoall implementation and the proposed API enhancements for KNL MCDRAM.

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] "OpenMP," `http://openmp.org/wp/`.
[3] MPI Forum, "MPI: A Message Passing Interface," in *www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf*.
[4] J. W. Poulsen, P. Berg, K. Raman, "Refactoring for Xeon Phi," https://coaps.fsu.edu/LOM/pdf/036_J_Weissman.pdf.
[5] Intel Xeon Phi Processor "Knights Landing" Architectural Overview, https://www.nersc.gov/assets/Uploads/KNL-ISC-2015-Workshop-Keynote.pdf.
[6] OpenSHMEM Standard (v1.1), http://openshmem.org/.

[7] Cray, Inc., “Man Page Collection: Cray Message Passing Toolkit.”

[8] K. Vaidyanathan, D. Kalamkar, K. Pamnany, J. Hammond, P. Balaji, D. Das, J. Park, and B. Joó, “Improving Concurrency and Asynchrony in Multithreaded MPI Applications Using Software Offloading,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’15. New York, NY, USA: ACM, 2015, pp. 30:1–30:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807602

[9] 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.

[10] 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.

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

[12] P. J. Mendygral, T. W. Jones, and K. Dolag, “MHD Simulations of Active Galactic Nucleus Jets in a Dynamic Galaxy Cluster Medium,” *Astrophysical Journal*, vol. 750, p. 166, May 2012.

[13] D. Holmes, A. Skjellum, M. Farmer, P. Bangalore, “Persistent Collective Operations in MPI,” http://www.epigram-project.eu/wp-content/uploads/2015/07/Holmes-ExaMPI15.pdf.

[14] The Ohio State University, “OSU MPI Benchmarks,” http://mvapich.cse.ohio-state.edu.

[15] 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

[16] 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

[17] 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.

[18] 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