Investigating High Performance RMA Interfaces for the MPI-3 Standard

Vinod Tipparaju, *Oak Ridge National Laboratory*
William Gropp, *University of Illinois*
Hubert Ritzdorf, *NEC Europe Ltd*
Rajeev Thakur, *Argonne National Laboratory*
Jesper L. Träff, *NEC Europe Ltd*

***Abstract*—The MPI-2 Standard, released in 1997, defined an interface for one-sided communication, also known as remote memory access (RMA). It was designed with the goal that it should permit efficient implementations on multiple platforms and networking technologies, and also in heterogeneous environments and non-cache-coherent systems. Nonetheless, even 12 years after its existence, the MPI-2 RMA interface remains scarcely used for a number of reasons. This paper discusses the limitations of the MPI-2 RMA specification, outlines the goals and requirements for a new RMA API that would better meet the needs of both users and implementers, and presents a strawman proposal for such an API. We also study the tradeoffs facing the design of this new API and discuss how it may be implemented efficiently on both cache-coherent and non-cache-coherent systems.**

I. Introduction

One-sided communication or remote memory access (RMA) is a promising paradigm for high-performance communication on low-latency networks and over shared memory. The main advantage of RMA lies in its asynchronous nature: Unlike in point-to-point (or two-sided) communication where the sender and receiver explicitly call send and receive functions, in RMA only the origin process calls the data-transfer function (put or get), and data transfer takes place without the target process explicitly calling any function to transfer the data. This model allows parallel programs to be less synchronizing and allows communication hardware to move data from one process to another with maximal efficiency. In addition, by eliminating the "tag matching" that is a key part of the send-receive model, RMA offers the promise of lower latency for short data transfers than the two-sided approach.

Because of the growing popularity of RMA, the MPI Forum defined a specification for RMA as part of the MPI-2 standard [1]. MPI defines three data-transfer functions for RMA: put (remote write), get (remote read), and accumulate (remote update). These functions must be used in conjunction with any of three synchronization methods, as shown in Figure 1. The synchronization methods enable the target process to indicate when its memory is ready for being read or written by a remote process, and they also specify when the data transfer is completed. The first two synchronization methods in Figure 1 are known as active-target synchronization because the target processes must participate in the synchronization. The third method is known as passive-target synchronization because the target does not need to call any function.

Process 0	*Process 1*
`MPI_Win_fence(win)`	`MPI_Win_fence(win)`
`MPI_Put(1)`	`MPI_Put(0)`
`MPI_Get(1)`	`MPI_Get(0)`
`MPI_Win_fence(win)`	`MPI_Win_fence(win)`

a. Fence synchronization

Process 0	*Process 1*	*Process 2*
	`MPI_Win_post(0,2)`	
`MPI_Win_start(1)`		`MPI_Win_start(1)`
`MPI_Put(1)`		`MPI_Put(1)`
`MPI_Get(1)`		`MPI_Get(1)`
`MPI_Win_complete(1)`		`MPI_Win_complete(1)`
	`MPI_Win_wait(0,2)`	

b. Post-start-complete-wait synchronization

Process 0	*Process 1*	*Process 2*
`MPI_Win_create(&win)`	`MPI_Win_create(&win)`	`MPI_Win_create(&win)`
`MPI_Win_lock(shared,1)`		`MPI_Win_lock(shared,1)`
`MPI_Put(1)`		`MPI_Put(1)`
`MPI_Get(1)`		`MPI_Get(1)`
`MPI_Win_unlock(1)`		`MPI_Win_unlock(1)`
`MPI_Win_free(&win)`	`MPI_Win_free(&win)`	`MPI_Win_free(&win)`

c. Lock-unlock synchronization

Figure 1. The three synchronization methods for one-sided communication in MPI-2. The numerical arguments indicate the target rank.

Many MPI implementations support RMA, with varying levels of optimization [2]–[10]. Nonetheless, even 12 years after its existence in the MPI standard, the MPI RMA functionality is rarely used in applications (with some notable exceptions such as [11]). One reason is that the synchronization methods, although needed in a programming model, add overhead to the basic data transfer functions. Other reasons include differences between what the MPI RMA model supports and what users seem to expect an RMA model to do [12], [13].

The MPI-3 standard, currently under development by the MPI Forum, provides an opportunity to address the limitations of the MPI-2 RMA interface in a way that better meets the needs of both users and implementers. In this paper, we examine the reasons for revisiting the MPI-2 RMA interface, present a strawman proposal that could be used as a starting point for defining a new interface in MPI-3, and discuss architectures that pose interesting challenges for implementing this interface.

II. Need for a Better RMA Interface

Partitioned Global Address Space (PGAS) languages such as UPC [14] and Co-Array Fortran [15] (under consideration as part of Fortran 2008) rely on efficient RMA operations. Library-based RMA approaches, such as SHMEM [16] and Global Arrays [17], have been used by a number of important applications. It is natural to look at the MPI-2 RMA interface as an implementation layer for these programming models. Doing so, however, has identified a number of mismatches between the MPI-2 RMA model and the needs of both PGAS languages and libraries such as Global Arrays.

A. Limitations of the MPI-2 RMA Model

In the last decade, several criticisms of the MPI-2 RMA model have been brought up by the user community. Some of the issues raised were in the context of PGAS languages [12], [13]. As described in Section I, MPI-2 RMA defines three synchronization modes. The passive target mode is more suitable for use as a compilation target for PGAS languages because of its truly one-sided nature. However, MPI-2 RMA requires collective creation (using `MPI_Win_create`) of memory that will participate in RMA operations. This is not possible in PGAS languages that permit all (or most) of memory to be accessed by remote memory operations. To enforce a precise correctness model, MPI-2 RMA makes overlapping RMA operations with Get and/or Put erroneous; PGAS languages make overlapping operations valid but undefined. MPI also has restrictions on the use of overlapping memory windows [12].

B. Memory Consistency Models

Users typically view remote memory accesses as extensions to load/store operations in a distributed memory environment. Thus, RMA is central to any Global Address Space (GAS) or Partitioned Global Address Space language or library. The (P)GAS languages and libraries typically define an abstract memory model that represents a shared-memory machine. This memory model defines the consistency guarantees for memory accesses. This means that despite the actual characteristics of the underlying system, memory, and network, these consistency guarantees must be met. Typically an RMA interface used by the (P)GAS languages and libraries is expected to enforce the consistency guarantees that are required for memory access. The challenge for a low-level RMA interface is to provide an efficient match to both the consistency models that are efficiently implemented by the system and the consistency models expected in the programming models.

MPI-2 selected an RMA model that could be implemented efficiently on a wide range of platforms (as discussed in Section III-B2). However, this model is not a good match to PGAS and similar programming models. Similarly, the consistency models used in PGAS languages do not fit all fast parallel systems. Hence it is important for an RMA interface to allow for enforcement of a variety of consistency models in its memory access. Some of the memory consistency models that are used in (P)GAS languages distinguish memory accesses by category of access. For example, a language may distinguish between sections of the code that need strict consistency from its remote memory access and sections of the code that could do with weak consistency. Such usage warrants flexibility from an RMA interface. However, working around different memory models and hybrid architectures that have a vaguely defined memory model and yet give consistency guarantees to a memory access is not a trivial task. For example, on a machine that is not cache coherent, ensuring write consistency for remote accesses is not trivial.

III. Constraints on a Solution

The limitations of the MPI-2 RMA model and users' perception of an RMA model have been described. However, many of these limitations originally were the result of an attempt to guarantee a uniform definition of correctness on various architectures. These issues need to be revisited with current and upcoming architectures in mind and based on what users have come to expect from a RMA model.

When defining an RMA interface, several features must be considered. These include: what is the consistency model for remote memory operations, what are the atomicity and granularity properties of access, and how is completion of data transfer indicated at both the source and destination. The following sections describe some of the choices and illustrate the difficulty in picking one particular choice of consistency model or completion approach.

A. Memory Model Issues and Properties of Remote Memory Access

1) Consistency and Completion: Shared memory consistency models are a deep and complex topic. Consistency for RMA should be defined as it would be for a shared memory access. Here, we merely discuss some of the different memory consistency models that elucidate desirable properties of a remote memory access.

Read/Write Consistency. One of the desired properties of RMA with respect to a single source (e.g., a single MPI process) is the guarantee that any value written by the source to a memory can be observed by a subsequent read from the same source, when the destination of the write operation has not been altered by writes from other sources (e.g., other MPI processes). This property cannot be guaranteed even with a shared memory access on a non-cache-coherent machine. We refer to this as the *ordering* property of an RMA operation.

Causal Consistency. Accesses to memory that are causally unrelated can occur or be observed in any order in the Causal Consistency model [18]. However, a particular order has to be agreed among causally related accesses.

For unrelated accesses, the RMA interface should allow for *unrestricted*, high-performance, remote memory access (see Section IV). In order to agree upon a particular order and enforce it, interfaces for synchronizations, fences, and verification of *remote completion* of a RMA operation will be necessary.

Sequential Consistency. This is a model with strong semantics that was defined by Lamport [19] as:

> the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

This means that it is not merely sufficient to ensure that the memory accesses are ordered with respect to a single source. In addition, multiple, potentially contending, accesses from different sources must be serialized. Each access at the destination must occur exclusively. RMA with *atomicity* property can achieve this effect. The HPC language community is reluctant to require sequential consistency due to its potential performance cost.

Hybrid Consistency. Some models such as Location Consistency [20] and RAO [21] describe the assumptions from a memory model (they expect the memory access layer to satisfy some of these assumptions) and define a hybrid consistency [22] model that benefits from weaker consistency for some accesses and strict for others. Originally proposed by Gao and Sarkar, location consistency does not rely on the memory coherence assumption. Instead, the state of a memory location is modeled as a partially ordered multiset of write and synchronization operations. The Relaxed Atomic + Order (RAO) model proposed by Saraswat et al. is another such model. These hybrid models can benefit from all the three above mentioned properties (*ordering, atomicity* and *remote completion*). For example, for accesses with weaker consistency, ordering property may suffice; but accesses with strict consistency will rely on the atomicity property.

2) Granularity: Remote memory accesses at the granularity of a word match well to a shared memory load/store style access. However, remote memory accesses have higher latencies than shared memory accesses. Hence, RMA is often used to transfer sections of remote memory. Although consistency is typically defined for accesses with the granularity of a word, the desired properties of RMA derived from the consistency requirement apply to RMA for sections of memory as well. Similarly, completion semantics (such as remote completion) bear the same meaning, whether the access is for a word or for a section of remote memory.

Given this diversity of RMA models, it makes sense to consider a more flexible RMA interface that permits the user to specify the required features (which we call *attributes*) of the RMA model. In the next section, we review the hardware (architectural) diversity that a successful RMA standard such as MPI must use effectively.

B. Architectural Issues

A remote memory access interface must work effectively with a wide range of architectures. Many of the fastest systems make compromises in the memory system that trade simplicity in the memory model (such as strict sequential consistency) with speed. These features vary from lack of cache coherence to the presence of vector and other special-purpose processing elements. It is important to consider these issues.

In discussing various architectures and how their memory models pose potential challenges in implementing RMA, we often refer to a *Serializer*. The serializer is in essence a mechanism to execute memory access operations on a remote address space in sequence, as described by Arvind and Maessen [23]. To enforce the *atomicity* attribute, a serializer such as this is essential. However, it may not be possible to implement a serializer efficiently on all architectures.

RMA attributes such as *ordering* and *remote completion*, when they are offered as features by the underlying network, are trivial to implement. When a network offers a mechanism to check for remote completion but doesn't guarantee ordering of data transfers, the ordering attribute can still be be guaranteed with a slight penalty. Similarly, on a network with no direct mechanism to check for remote completion but with message ordering as a natural property, remote completion may be guaranteed with a slight penalty. However, on systems with networks that do not have methods to check for remote completion or message ordering property, additional software mechanisms may be required even to allow for *ordering* and *remote completion* attributes.

1) Cache-Coherent Systems: On cache-coherent systems, coherence exists between cache lines on CPU's, any shared cache, network, and main memory. Machines of this type represent most of the Top 500 systems today. The recently announced 20 PetaFLOP IBM Blue Gene/Q system, Sequoia, will also have these characteristics.

On a Cache-Coherent system, RMA communication may directly use the fastest and the lowest level communication library available for the system. When ordering is not guaranteed by the underlying network (e.g., the Quadrics QSNetII and QSNetIII networks), additional software support (e.g., a counter for messages or software epochs) may be required to enforce ordering. Typically, a communication thread has been used in implementations that allow for atomic updates of sections of remote memory. This communication thread could be implicit or explicit. An example of an implicit communication thread is the handler of an active message. Examples of explicit communication threads are the communication helpers used in RMA libraries. The use of such a thread ensures serialized handling of incoming

messages without the requirement of locks. However, implicit or explicit communication threads may not be available on some architectures.

An example of a cache-coherent system is the Cray XT(3/4/5). On the Cray XT at the Sandia National Laboratories, the Catamount kernel [24] does not allow users to spawn additional threads for communication, and the Portals communication library [25] on the Cray XT does not support Active Messages. Thus, to implement atomicity, a coarse-grained lock is required. The use of these mechanisms results in low performance for the RMA calls that set the *atomicity* attribute. However, if the Compute Node Linux [26] is used on the XT system, communication threads may still be used along with the Portals library in the implementation.

Some shared address space architectures with scalar and vector processing elements, such as the Cray X1E, demonstrate unique cache properties. On the X1E, local (same node) accesses cache data and maintain coherence, whereas remote accesses are not cached. Since coherence exists for the accesses within a node, it does not pose any additional challenges in implementing RMA operations.

2) Non-Cache-Coherent Systems: Due to the complexity and cost of maintaining cache coherence across many processors and/or over a network, some high-performance systems are designed to not provide (full) cache coherency. In addition, for similar reasons, some architects believe that full cache coherency may not be supported on multicore chips should the number of cores become very large. Examples of non-cache coherent systems are the NEC SX machines.

The NEC SX series are clustered vector systems. The processor consists of a vector and a scalar unit. The vector unit accesses the banked memory directly, while the scalar unit uses a write-through cache to hide memory latencies. For the SX-9, each node has up to 16 processors, sharing the same memory via a memory network. Nodes are connected through a proprietary, high-performance switch. The scalar caches are not coherent, neither between processors on different nodes, nor between processors on the same node. The memory adheres to a relaxed consistency model, and a memory fence operation is needed to ensure that previous scalar and vector accesses to memory have completed. Since data in cache may have been invalidated by a write by another processor (whether on the same node or on a different node), it may be necessary to clear the cache or to circumvent the cache by reading (with vector instructions) directly from memory. Processors inform other processors of memory writes by write-fence operations (setting a flag) or by atomic operations. For RMA, this implies that involvement of the target is needed to either invalidate caches or otherwise make the process aware of data written by other processes. Note that MPI-2's RMA model was designed to support systems like the SX series.

3) Hybrid Systems with Special Purpose Processing Elements: Systems with accelerators and special purpose processing elements (PE) are not new to the HPC community. In the recent past, systems with special purpose processing elements such as the Road Runner [27] system at the Los Alamos National Laboratory have emerged and continue to do so. A special purpose PE can both be realized as an MPI task or as an attached device to an MPI task. If the special purpose PE (such as an accelerator, FPGA, GPU, or Cell) were seen as an attached device to an MPI task running on general purpose CPU, the model used to program and access that device is orthogonal to MPI and its semantics. However, when the special purpose processor is seen as an MPI task, MPI communication to and from the special purpose process will have to work with the same RMA attributes. However, given the nature of the special purpose PE's (such as the GPU), there are some issues that need special consideration.

Address space and virtual pointer: The address space in the special purpose processor may be of a different size than that of the rest of the system. Interface designers have to account for this when accessing remote memory.

Endianness: The special purpose PE may have a different endianness. If the architecture doesn't allow for switchable endianness, this may pose additional challenges in data transfers, more so for RMA because of the one-sided nature of the operations. For example, a system built from IBM's POWER architecture and commodity GPUs will most likely have different endianness for data items.

IV. A Strawman Proposal

In the previous sections, we have described some of the constraints on an RMA model, both in terms of the software (programming model) and hardware (RMA support in the architecture). This leads to a number of design requirements:

1) In order to support RMA to arbitrary locations, no constraints on memory, such as symmetric allocation or collective window creation, can be permitted.
2) To allow for overlap of communication with other operations, nonblocking RMA operations are required.
3) RMA operations that are imprecise (such as access to overlapping storage) should be permitted, even if the behavior is undefined.
4) To permit low-latency operations, RMA operations should be possible in a single routine call (these would be blocking RMA operations).
5) A user must be able to specify the required level of consistency, atomicity, and completeness, and the interface should accommodate that. In addition, it should be relatively easy to change those requirements (permitting the use of the most stringent rules while debugging).
6) The RMA model must support non-cache-coherent and heterogeneous environments.
7) Transfers of noncontiguous data, including strided (vector) and scatter/gather must be supported.

8) Scalable completion (a single call for a group of processes) is required.

These requirements suggest an interface that makes use of existing MPI concepts such as communicators for groups of processes, datatypes for heterogeneity and noncontiguous data, and requests for completion of nonblocking operations. With these pieces, the key operation becomes

```
MPI_RMA_put(origin_addr, origin_count,
    origin_datatype, target_mem, target_disp,
    target_count, target_datatype,
    target_rank, comm, RMA_Attributes,
    request)
```

IN	origin_addr	The starting (local) address of the put
IN	origin_count	the number of entries
IN	origin_datatype	datatype of each entry in origin buffer
IN	target_mem	object representing the target memory being accessed
IN	target_disp	displacement from start of target buffer represented by target_mem
IN	target_count	number of entries in target buffer
IN	target_datatype	datatype of each entry in target buffer
IN	target_rank	rank of target
IN	comm	communicator
IN	rma_attributes	the attributes of this RMA operation
OUT	request	communication request

In this operation, the `rma_attributes` parameter gives the user the flexibility of specifying the attributes derived in Section III-A: *ordering, remote completion*, and *atomicity*. In addition to RMA operations with configurable attributes, blocking RMA operations need to be supported. An additional attribute, *blocking*, can be used to achieve this. By setting the blocking attribute, the user can do single call RMA updates. The user may choose to set the attributes either at the level of a communicator or on a per-call basis. Also, the type representation of `target_mem` in this interface needs thorough consideration; the address space from which this call is made, i.e. the address space of the `origin_rank`, may be different from that of the `target_rank`. For example, the origin may be in a 32-bit address space where as the target may require 64-bit addressing (see Section III-B3). Hence the target needs to be represented by an object that can encompass this information (`target_mem` is further discussed in Section V). Finally, the request parameter in the interface may be used to check for completion of the RMA (using `MPI_Wait`, `MPI_Test`, and variants) for the calls without the blocking attribute. If the remote completion attribute is set, the completion is remote, otherwise it is local.

An interface for RMA Get operation follows. In both cases, origin represents the initiator of the call (hence `origin_rank` is the rank of a process that initiates the Put or Get) and `target` represents the process whose memory is being accessed remotely.

```
MPI_RMA_get(origin_addr, origin_count,
    origin_datatype, target_mem, target_disp,
    target_count, target_datatype,
    target_rank, comm, RMA_Attributes,
    request)
```

Similar operations for an RMA Accumulate may also be provided. However, since all three of the Get, Put, and Accumulate RMA operations have similar semantics, a single interface to represent all three of the above operations is an alternative.

```
MPI_RMA_xfer(rma_optype,accumulate_optype,
    origin_addr, origin_count,
    origin_datatype, target_mem, target_disp,
    target_count, target_datatype,
    target_rank, comm, RMA_Attributes,
    request)
```

In this operation, the `rma_optype` can be used indicate whether this call is a Put, a Get or an Accumulate. The advantage of such an optype to represent this call is that in the future, this optype may be used for expanding the interface. One example of such expansion is the invocation of a remote function (a remote method invocation) or signaling a remote thread.

The interface for `MPI_RMA_xfer` lets the user set the remote completion attribute and verify remote completion of each individual RMA. A useful, less restrictive alternative is to not set the remote completion attribute for each individual RMA and yet allow for checking remote completion of a subset of remote memory accesses to a particular destination.

```
MPI_RMA_complete(comm, target_rank)
```

IN	comm	the communicator
IN	target_rank	rank of the target of this completion

This operation allows us to check for remote completion of all previous remote memory accesses to a particular `target_rank`. However, to do this for every rank whose memory has been accessed may become a programming burden. To alleviate this, a value of `MPI_ALL_RANKS` for the `target_rank` can be used to indicate that the operation is to be done across all the ranks in the communicator. Using `MPI_ALL_RANKS` for `target_rank` allows us to replace:

```
for target_rank=first_rank through
    target_rank=last_rank in the communicator
     comm
  do MPI_RMA_complete(comm,target_rank)
```

with:

```
MPI_RMA_complete(comm,MPI_ALL_RANKS)
```

However, if all the processes in the communicator wanted to do this collectively, it may be possible to perform additional implementation optimizations with prior knowledge of the participation of remote processes in the complete call. Hence, an interface to do this collectively is useful.

```
MPI_RMA_complete_collective(comm)
```

`MPI_RMA_complete` and its collective version are strict synchronization operations. Additional weaker synchronization mechanisms for RMA will give programmers flexibility in the kind of synchronization they use. The ordering attribute allows for ordering between two RMA operations. This is a weaker form of synchronization than remote completion. However, instead of setting the ordering attribute for each individual RMA, the users may benefit from an operations that orders among sets of RMA operations (similar to `shmem_fence`).

```
MPI_RMA_order(comm, target_rank)
```

IN	comm	the communicator
IN	target_rank	rank of the target of this ordering

Similar to its usage in `MPI_RMA_complete`, `MPI_ALL_RANKS` may be used to order among sets of operations with all processes in the communicator with a single call. A collective version of this interface would be

```
MPI_RMA_order_collective(comm)
```

V. Strawman API Discussion

The strawman proposal attempts to address some of the limitations of the MPI-2 RMA model. Issues with `MPI_Win` and active target have been addressed. The object representing the target memory, `target_mem`, need not be allocated collectively. The user is responsible for passing the `target_mem` object to the MPI processes that need to access memory remotely. Collective allocation of `target_mem` and interfaces to associate existing user memory (heap/stack) to a `target_mem` object are currently being discussed and formulated. Concurrent Put/Get operations targeting overlapping memory regions are not defined as erroneous. There is no default restriction on concurrent updates to different target memory objects that may be representing overlapping regions in target memory. The user may choose to impose this restriction by configuring RMA attributes. Datatypes are supported, hence application developers and HPC language community can use these interfaces to transfer noncontiguous data.

Only a few configurable attributes are listed. A more thorough investigation of the application and the HPC language domain may bring out other attributes useful for RMA operations. Remote method invocation (RMI) or active message support is not included. This is primarily because supporting these operations requires a thorough investigation of the ability of current and upcoming networks and operating systems to support such a feature. On some architectures (such as the ones described in Section III-B3), it is not trivial to define correct behavior of such a feature. The MPI Forum has formed a working group to investigate active messages and RMI. Interfaces for allocating the object representing the target memory, `target_mem`, or interfaces for associating the `target_mem` object with user memory are not described in this paper. This requires a thorough analysis of operating system space and network interconnect requirements (the network interconnect may require the memory to be registered, for example). Read-Modify-Write (RMW) operations (fetch-and-increment, compare-and-swap etc.) are not described in this paper. RMW operations with semantics similar to RMA are being discussed in the MPI forum as a part of this strawman proposal. Two kinds of Read-modify-write operations, one for conditional RMW and other for unconditional RMW are being considered.

There are some issues that need to be addressed with respect to the implementation of these interfaces on different architectures. Cache coherence (or lack of it) poses interesting problems. Networks today do not have support for atomically accessing arbitrary sections of remote memory. However, the need for atomic updates exists in HPC languages and applications. Working around the lack of this support in networks today is a challenge and poses implementation overheads. Mechanisms such as a communication thread that acts as a serializer may be required to guarantee an *atomicity* attribute when the HPC system lacks support for this. This may impact the ability to provide an efficient MPI RMA on platforms that lack both threads and support for the necessary RMA operations.

A. Prototype Implementation

Efficient implementations of such an RMA model require mechanisms to selectively guarantee the attributes based on the environment. When the underlying architecture and network interconnect have mechanisms to guarantee these attributes, implementation is trivial. However, when such mechanisms are not available, guaranteeing these attributes may incur software overhead. The highest performance penalty will be incurred to enforce the atomicity guarantee. There are effective serializers such as additional execution contexts (additional thread or process) to achieve this. In the absence of an efficient serializer, a very coarse grain (MPI Process level) access control may serve as an alternative. In the absence of either of these mechanisms, one has to rely on MPI progress (with associated loss of efficiency).

We performed a few basic experiments on a Cray XT5 system. Cray MPT 3.1 was used, and the prototype RMA code was written using the Portals communication library. The Portals library on the Cray XT allows the user to check for remote completion of a message via an Event Queue mechanism. Our objective was to evaluate the performance of interfaces that give a choice of completion and consistency attributes. Hence we measure the communication cost associated with each individual attribute for an implementation that uses two different kinds of serializers: 1) a communication thread, and 2) coarse grain MPI process level access control.

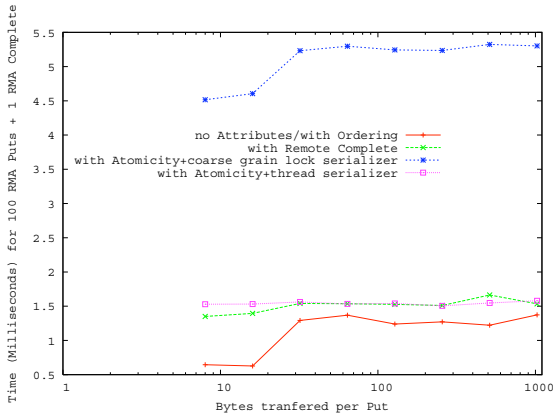

Figure 2. The cost of each attribute on the Cray XT5

In the experiment, seven MPI processes (one on each of the XT5 nodes) concurrently do 100 puts to overlapping memory regions on process 0, followed by a single RMA Complete call. The experiment does these puts first with no attributes, then with *ordering* set, followed by *remote completion* set, and finally with *atomicity* attribute. The *Blocking* attribute is always set in this example to use single call RMA update. Figure 2 shows the time taken for 100 RMA put operations and one RMA Complete operation for data sizes varying from 8 bytes to 1 kilobyte. Since message ordering is a natural property of the Cray XT5 network, the line representing *ordering* attribute overlaps the line representing the case with no attributes. The objective of this experiment is to show the best case behavior with no attributes and the worst case behavior with the atomicity attribute. Note that when a communication thread is used as a serializer, despite seven different processes doing the put operation to overlapping memory regions, they can be serialized and atomicity attribute enforced with low overhead. However, with a coarse grain serializer, such as the process level lock used in this example to implement atomicity attribute, there is a significant performance penalty.

VI. Related RMA API

The Aggregate Remote Memory Copy Interface (ARMCI) library [28] and GASNet [29] are two communication subsystems that support RMA styled communication API and are used by Global Address Space languages and libraries. ARMCI is used by the Global Arrays Toolkit [17]. GASNet is the communication layer for the Berkeley UPC compiler [14]. Suitability of ARMCI, GASNet and some of the other existing RMA interfaces for use with MPI is discussed in detail by Buntinas and Gropp [30].

ARMCI has support for contiguous, vector and strided RMA Put, Get and Accumulate operations. Both blocking and non-blocking versions of these operations are supported. All blocking operations are ordered by the library and all non-blocking operations have no ordering guarantee. Accumulate operations are serialized. MPI-2 standard allowed for all the reduce operations to be done as a part of the Accumulate operation. In ARMCI Accumulate only allows an operation similar to a *daxpy* where `x` is the remote memory and `y` and `a` are inputs to the accumulate operation. The primary addition that the strawman MPI-3 RMA API offers over the model supported by ARMCI is flexibility in the attributes of the RMA operation and more powerful completion semantics. For example, it is possible to have a blocking unordered RMA operation with the strawman API proposed here for MPI-3. It is also possible to check local or remote completion of a subset of RMA operations. Neither of these is possible with the current ARMCI API.

GASNet has a core API based on the Active Message paradigm. GASNet has different interfaces for short, medium and long active messages. No particular ordering is guaranteed for these operations nor is it possible to specify any. In addition to the core API, GASNet also specifies an extend API that supports RMA Put and Get operations. One of the difference between the strawman API proposed here and GASNet RMA API is the support for accumulate operation – GASNet API does not have explicit interfaces for doing the Accumulate operation. The MPI-3 RMA strawman interfaces described in this paper include the accumulate operation. Another difference is the support for non-contiguous data transfers – the current GASNet extend API RMA specification (version 1.8) does not include support for non-contiguous data transfers. The proposed strawman API for MPI-3 RMA allows for datatypes and hence allows for both heterogeneity and noncontiguous data.

VII. Conclusions

We have discussed the limitations of the existing MPI-2 RMA interface and the need for providing an interface that better meets the requirements of applications, libraries, and PGAS languages. We have presented a strawman proposal that incorporates many of the needed features. We note that the proposed interface is only an initial draft and not the final MPI-3 RMA design. We hope the proposal will help initiate a discussion in the broader community to better define the goals, scope, and design of an RMA interface in MPI-3. We invite feedback from vendors, application developers and other users interested in RMA communication.

References

[1] "MPI-2: Extensions to the message-passing interface." [Online]. Available: http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html

[2] N. Asai, T. Kentemich, and P. Lagier, "MPI-2 implementation on Fujitsu generic message passing kernel," in *Proc. of SC99: High Performance Networking and Computing*, November 1999.

[3] S. Booth and E. Mourão, "Single sided MPI implementations for SUN MPI," in *Proc. of SC2000: High Performance Networking and Computing*, November 2000.

[4] M. Golebiewski and J. L. Träff, "MPI-2 one-sided communications on a Giganet SMP cluster," in *Proc. of the 8th European PVM/MPI Users' Group Meeting*, Sep. 2001, pp. 16–23.

[5] W. Jiang, J. Liu, H.-W. Jin, D. K. Panda, D. Buntinas, R. Thakur, and W. Gropp, "Efficient implementation of MPI-2 passive one-sided communication over InfiniBand clusters," in *Proc. of the 11th European PVM/MPI Users' Group Meeting*, Sep. 2004, pp. 68–76.

[6] W. Jiang, J. Liu, H.-W. Jin, D. K. Panda, W. Gropp, and R. Thakur, "High performance MPI-2 one-sided communication over InfiniBand," in *Proc. of the 4th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGrid 2004)*, April 2004.

[7] E. Mourão and S. Booth, "Single sided communications in multi-protocol MPI," in *Proc. of the 7th European PVM/MPI Users' Group Meeting*, Sep. 2000, pp. 176–183.

[8] R. Thakur, W. Gropp, and B. Toonen, "Optimizing the synchronization operations in MPI one-sided communication," *International Journal of High-Performance Computing Applications*, vol. 19, no. 2, pp. 119–128, Summer 2005.

[9] J. L. Träff, H. Ritzdorf, and R. Hempel, "The implementation of MPI-2 one-sided communication for the NEC SX-5," in *Proc. of SC2000: High Performance Networking and Computing*, November 2000.

[10] J. Worringen, A. Gäer, and F. Reker, "Exploiting transparent remote memory access for non-contiguous and one-sided-communication," in *Proc. of the 2002 Workshop on Communication Architecture for Clusters*, April 2002.

[11] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and Y. Kaneda, "16.4-TFLOPS direct numerical simulation of turbulence by a Fourier spectral method on the Earth Simulator," in *Supercomputing*, 2002.

[12] D. Bonachea, "The inadequacy of the MPI 2.0 one-sided communication API for implementing parallel global address-space languages," http://www.cs.berkeley.edu/~bonachea/upc/mpi2.html.

[13] D. Bonachea and J. Duell, "Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations," in *2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing*, 2003.

[14] "UPC language specifications, v1.2," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-59208, 2005.

[15] R. W. Numrich and J. K. Reid, "Co-array Fortran for parallel programming," *ACM Fortran Forum*, vol. 17, no. 2, 1998.

[16] R. Barriuso and A. Knies, "Shmem user's guide for c," Cray Research, Inc, Tech. Rep., 1994.

[17] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global Arrays: A non-uniform-memory-access programming model for high-performance computers," *The Journal of Supercomputing*, vol. 10, no. 2, pp. 169–189, 1996.

[18] P. Hutto and M. Ahamad, "Slow memory: weakening consistency to enhance concurrency in distributed shared memories," *Distributed Computing Systems, 1990. Proceedings., 10th International Conference on*, pp. 302–309, May-1 Jun 1990.

[19] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess progranm," *IEEE Trans. Comput.*, vol. 28, no. 9, pp. 690–691, 1979.

[20] G. Gao and V. Sarkar, "Location consistency-a new memory model and cache consistency protocol," *Computers, IEEE Transactions on*, vol. 49, no. 8, pp. 798–813, Aug 2000.

[21] V. A. Saraswat, R. Jagadeesan, M. Michael, and C. von Praun, "A theory of memory models," in *PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 2007, pp. 161–172.

[22] H. Attiya and R. Friedman, "A correctness condition for high-performance multiprocessors (extended abstract)," in *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1992, pp. 679–690.

[23] A. Arvind and J.-W. Maessen, "Memory model = instruction reordering + store atomicity," *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on*, pp. 29–40, 2006.

[24] S. M. Kelly and R. Brightwell, "Software architecture of the light weight kernel, catamount," in *in Proceedings of the Cray User Group Annual Technical Conference*, 2005.

[25] R. Brightwell, R. Riesen, B. Lawry, and A. Maccabe, "Portals 3.0: Protocol building blocks for low overhead communication," *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pp. 164–173, 2002.

[26] D. Wallace, "Compute node linux: New frontiers in compute node operating systems," in *in Proceedings of the Cray User Group Annual Technical Conference*, 2007.

[27] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the petaflop era: The architecture and performance of roadrunner," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–11.

[28] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. K. Panda, "High Performance Remote Memory Access Communication: The Armci Approach," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 233–253, 2006.

[29] D. Bonachea, "Gasnet specification, v1.1," Berkeley, CA, USA, Tech. Rep., 2002.

[30] D. Buntinas and W. Gropp, "Understanding the requirements imposed by programming model middlware on a common communication subsystem," http://www.cs.uiuc.edu/homes/wgropp/bib/papers/2005/ccs-requirements.pdf.