Building Scalable PGAS Communication Subsystem on Blue Gene/Q

Abhinav Vishnu #1 Darren Kerbyson #2 Kevin Barker #3 Hubertus van Dam #4

#1,2,3 *Performance and Architecture Lab, Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, WA 99352*
#4 *Computational Chemistry Group, Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, WA 99352*
[1] abhinav.vishnu@pnnl.gov
[2] darren.kerbyson@pnnl.gov
[3] kevin.barker@pnnl.gov
[4] hubertus.vandam@pnnl.gov

***Abstract*—This paper presents a design of scalable Partitioned Global Address Space (PGAS) communication subsystems on recently proposed Blue Gene/Q architecture. The proposed design provides an in-depth modeling of communication infrastructure using Parallel Active Messaging Interface (PAMI). The communication infrastructure is used to design time-space efficient communication protocols for frequently used data-types (contiguous, uniformly non-contiguous) with Remote Direct Memory Access (RDMA) get/put primitives. The proposed design accelerates load balance counters by using asynchronous threads, which are required due to the missing network hardware support for generic Atomic Memory Operations (AMOs). Under the proposed design, the synchronization traffic is reduced by tracking conflicting memory accesses in distributed memory with a slight increment in space complexity. An evaluation with simple communication benchmarks show a adjacent node get latency of 2.89μs and peak bandwidth of 1775 MB/s resulting in $\approx$ 99% communication efficiency.The evaluation shows a reduction in the execution time by up to 30% for NWChem self consistent field calculation on 4096 processes using the proposed asynchronous thread based design.**

I. Introduction

IBM Blue Gene/Q is a recently proposed system-on-a-chip architecture with a potential for building supercomputers with a peak performance of tens of PetaFlop/s (PF/s) [1], [2]. At this massive scale, revisiting the programming models, which can effectively leverage the architecture is essential. An exploration of alternative/complimentary programming models to the ubiquitous Message Passing Interface (MPI) [3], [4] is being undertaken by Partitioned Global Address Space (PGAS) models such as Global Arrays [5], Unified Parallel C (UPC) [6], Co-Array Fortran [7], IBM X10 [8] and Chapel [9]. PGAS models rely on scalable distribution of frequently used data structures (arrays/trees), and asynchronous read/writes (get/put) to these structures for load balancing, work-stealing and resiliency [10]. An essential component of the PGAS models is the underlying communication infrastructure. The communication infrastructure provides abstractions for remote memory access (RMA), active messages (AM), atomic memory operations (AMOs), and synchronization. The communication runtime must harness the architectural capabilities and address the hardware limitations to scale the PGAS models.

This paper presents a design of scalable PGAS communication subsystem on Blue Gene/Q. The proposed framework involves designing time-space efficient communication protocols for frequently used datatypes such as contiguous, uniformly non-contiguous with Remote Direct Memory Access (RDMA). The proposed design accelerates the load balance counters, frequently used in applications such as NWChem [11] using asynchronous threads. The design space involves alleviating unnecessary synchronization by setting up status bit for memory regions. An implementation and evaluation of the proposed design with Aggregate Remote Memory Copy Interface (ARMCI) [12] shows a get latency of 2.89 μs and bandwidth of 1775 MB/s resulting in $\approx$ 99% communication efficiency. The load balance counters, accelerated using asynchronous thread based design reduce the execution time of NWChem [11] by up to 30% using Self Consistent Field (SCF) theory on 4096 processes.

A. Contributions:

Specifically, the contributions of the paper are:

- Detailed time-space complexity models of PGAS communication systems using Parallel Active Messaging Interface (PAMI) [13]. The paper contributes algorithms for datatypes (contiguous, uniformly non-contiguous) get/put communication and efficient handling of conflicting memory accesses, while providing location consistency [14].
- Application driven use-case to show that RDMA is necessary, but not sufficient for PGAS models. An asynchronous thread based design is presented to address the hardware limitations for load balance counters.
- An in-depth performance evaluation and analysis using communication benchmarks with datatypes, load/store (get/put) operations, atomic memory operations using up to 4096 processes. Performance analysis with NWChem [11] using Self Consistent Field (SCF) highlights a need for asynchronous threads for load balancing, and a need for network hardware support in future Blue Gene architecture.

The rest of the paper is organized as follows: Section II provides a background of our work. Section III provides a solution space for designing scalable PGAS communication subsystem, time-space tradeoff of communication protocols for datatypes, accelerating atomic memory operations and implementation details. Section IV provides a performance evaluation of the proposed design using communication benchmarks, and NWChem [11], a high performance computation chemistry application designed using Global Arrays [5]. Section V provides related work on designing scalable communication subsystems.

II. Background

This section provides a background for designing scalable PGAS communication subsystem for Blue Gene/Q. A description of Blue Gene/Q [1] and ARMCI [12] is provided as follows.

A. Blue Gene/Q Architecture

Blue Gene/Q is the third supercomputer design in the Blue Gene series. It continues to expand and enhance the Blue Gene/L and /P architectures. The Blue Gene/Q Compute chip is an 18 core chip. The 64-bit PowerPC A2 processor cores are 4-way simultaneously multi-threaded (SMT), and run at 1.6 GHz. Each processor core has a SIMD Quad-vector double precision floating point unit, the QPU, after which the system is named. The processor cores are linked by a crossbar switch to a 32 MB eDRAM L2 cache. The L2 cache is multi-versioned, supports transactional memory and speculative execution, and has hardware support for atomic operations [2]. L2 cache misses are handled by two built-in DDR3 memory running at 1.33 GHz [1], [2]. 16 Processor cores are used for computing, and a 17th core for operating system assist functions such as interrupts, and asynchronous I/O. The 18th core is fused out.

Blue Gene/Q Compute nodes are interconnected using a 5D torus configuration with ten 2 GB/s bidirectional links. 5D torus increases the bisection bandwidth in comparison to Blue Gene/P, increases the partitioning capacity and reduces the system diameter. Barrier and Collective communication network are integrated with the torus network. Blue Gene/Q supports deterministic and dynamic routing. However, the software interfaces at the point of paper submission support deterministic routing only. Blue Gene/Q provides rich software primitives for Active Messages and Remote Memory Access (RMA) model using Parallel Active Messaging Interface (PAMI). PAMI supports primitives for client creation, context creation, end-point creation, memory regions for Remote Direct Memory Access (RDMA), contiguous put/get and read-modify-write operations. These primitives are used in designing scalable PGAS communication subsystem on Blue Gene/Q architecture.

B. Aggregate Remote Memory Copy Interface

ARMCI [12] is a scalable communication runtime system which provides communication interfaces for remote memory access models. ARMCI provides communication primitives for data-types (contiguous, uniformly non-contiguous and general I/O vector), atomic memory operations (AMOs) and synchronization. It provides primitives for collective/non-collective memory allocation, atomic memory operations (fetch-and-add, compare-and-swap), bulk accumulate operations and lock/unlock operations. ARMCI supports the location consistency model [14]. The primitives support pairwise and collective memory synchronization. Support for non-blocking communication interfaces with explicit and implicit handles are supported, which follow buffer reuse semantics similar to MPI [3], [4].

The Global Arrays programming model provides abstractions for distributed arrays and leverages the communication primitives provided by ARMCI. Global Arrays programming model has been used for designing many scalable applications in domains such as chemistry [11] and sub-surface modeling [15]. ARMCI leverages the low level network primitives provided by modern networks and multi-core systems. It is supported on clusters with commodity interconnects (InfiniBand [16], Ethernet) and high-end systems (IBM BG/P, Cray XTs, Cray XE6/XK6 [17], [18]). This paper primarily focusses on designing a scalable PGAS communication layer for Blue Gene/Q systems using ARMCI.

III. Scalable ARMCI Design on Blue Gene/Q

This section begins with a brief overview of Parallel Active Messaging Interface (PAMI). This is followed by a discussion of preliminaries: time-space utilization of scalable communication protocols for ARMCI using PAMI communication library. This is followed by a detailed description of scalable communication protocols for load/store (get/put) primitives with data-types (contiguous, uniformly non-contiguous). The subsequent section makes a strong case for using asynchronous threads. The section concludes with important implementation details to reduce network synchronization traffic in order to maintain location consistency [14].

A. Overview of IBM Parallel Active Messaging Interface (PAMI)

IBM Parallel Active Messaging Interface (PAMI) is a recently proposed communication interface for Active Messaging and Remote Memory Access (RMA) primitives on Blue Gene/Q. PAMI interface is general and available for PERCS and x86 architectures as well.

1) Fundamentals: Figure 1 shows PAMI terminology with an example. A process must create a unique communication client for allocating network resources. A client must create at least one communication context. PAMI contexts

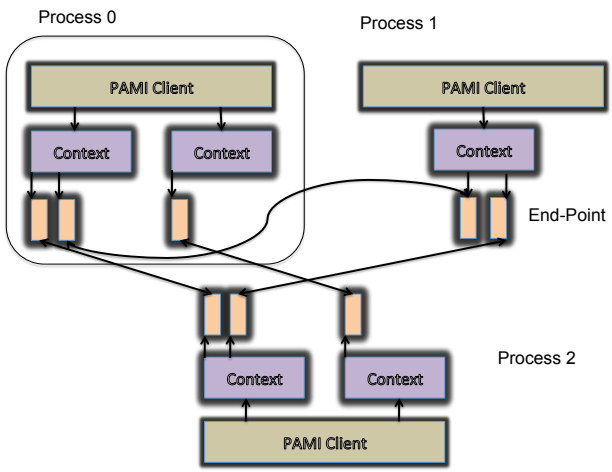

Figure 1. An Example of PAMI Setup with 3 processes: Process 0 and 2 have two communication contexts, Process 1 has only one context

are threading points, which may be used to optimize concurrent communication operations - multiple communication contexts may progress independently. A communication context must create end-points - objects which are used to address a destination in a client. A local endpoint must be created before initiating an active message or put request. A remote endpoint is required for get and atomic memory operations (AMOs).

2) Active Messaging: PAMI provides interfaces for non-blocking active messages with a support for local and remote call-backs. Active messaging interfaces are non-blocking and execute a local call-back for notifying completion. An exception is PAMI immediate variant of active message, which has blocking completion semantics.

3) Remote Memory Access (RMA): PAMI supports RMA model by providing remote load/store (get/put) for contiguous data transfer, AMOs (add, fetch-and-add, compare-and-test) and primitives for synchronizing the communication end-points. The RMA model may use Remote Direct Memory Access (RDMA) by leveraging memory region(s). PAMI RMA operations are non-blocking; a local/remote call-back function is required for notifying local/remote completion, respectively.

4) Ordering and Completion Semantics: By default, PAMI communication primitives provide message ordering between a pair of processes[1]. This is useful, but not required for location consistency semantics maintained by ARMCI [14]. An exception to this rule is atomic memory operations (AMOs), which are not ordered with respect to each other and with other messages in flight.

[1]This is true for dimension order routing, which is enabled by default on Blue Gene/Q architecture

B. Preliminaries

This section focusses on time-space complexity of designing ARMCI on Blue Gene/Q. Table I shows the attributes used in the following sections to model the space and time complexity of setting up ARMCI on Blue Gene/Q using PAMI. These attributes are used to derive the space utilized by data structures and space-time tradeoff(s) of communication protocols for RMA primitives.

	Property	Symbol
1	Message Size for Data Transfer	m
2	Total Number of Processes	p
3	Number of Processes/Node	c
4	Endpoint Space Utilization	α
5	Endpoint Creation Time	β
6	Memory Region Space Utilization	γ
7	Memory Region Creation Time	δ
8	Context Space Utilization	ϵ
9	Context Creation Time	ϱ
10	Number of Contexts	ρ
11	Communication Clique	ζ
12	Number of Active Global Address Structure	σ
13	Number of Local Buffers used for Communication	τ

Table I
PAMI TIME AND SPACE ATTRIBUTES

PAMI communication context(s) must be created before creating a communication endpoint for RMA/Active Messaging request. Space (M_c) and time (T_c) complexity/process is:

$$M_c = \epsilon.\rho \tag{1}$$

$$T_c = \varrho.\rho \tag{2}$$

The empirical value of ρ is typically 1. In the later sections, a case is presented, when using more contexts may be beneficial for accelerating PGAS communication.

A communication endpoint may be created as the communication clique (ζ) is generated during the lifetime of an application. Endpoint creation is local - active message and put communication primitives require only a local endpoint for initiating a data transfer. The local endpoint space utilization (M_e) and time complexity (T_e) for communication clique ζ is:

$$M_e = \zeta.\alpha.\rho \tag{3}$$

$$T_e = \zeta.\beta.\rho \tag{4}$$

The get communication primitive requires a remote endpoint. Hence, a space-time tradeoff exists in designing get primitive protocols: A source may use an active message to send the local address to a remote process, and the remote process performs a put to complete the get operation. This protocol alleviates a requirement for endpoint creation. However, it requires an involvement from remote process to complete the get operation. The proposed implementation caches endpoints for the communication clique increasing

the space utilization M_e, similar to previously proposed approaches for other Interconnects [16].

Blue Gene/Q supports Remote Direct Memory Access (RDMA). RDMA operations require that a memory region is created before using the region as a source/target of communication. The size of the meta-data for memory region is independent of the size of memory region. The space complexity(M_r) and time complexity (T_r) is:

$$M_r = \tau.\gamma + \sigma.\zeta.\gamma \tag{5}$$

$$T_r = \tau.\delta + \sigma.\delta \tag{6}$$

The space complexity M_r includes the memory required for caching the memory regions for communication clique and number of local buffers used for communication. With strong scaling, $\zeta \approx p$, which is prohibitive on a memory limited architecture like Blue Gene/Q. The proposed implementation addresses the memory limitation by using a remote memory region cache. Missing cache entries are served using an active message to the destination; the replacement policy uses least frequently used (LFU) algorithm.

The next section presents a design for communication protocols, which are optimized for a data-types (contiguous/strided), load/store (get/put) requests, atomic memory operations and synchronization.

C. Remote Memory Access Primitives

1) Contiguous Data-type: RDMA requires the source and target of RDMA to be contiguous in memory. Consequently, the contiguous data-type communication uses RDMA, when possible. An associated memory region cache is searched for finding local, remote memory regions, base addresses, and offsets for RDMA transfer.

At scale, the creation of memory region may fail due to memory constraints. The proposed design uses a fall-back protocol with active messages for contiguous data-type. Using the popular LogGP [19] model for communication (o: time during which processor is busy in communication, L: network latency, G: inverse of bandwidth) , the latency for get (T_{rdma}) operation using RDMA and fall-back ($T_{fallback}$) is defined as[2]:

$$T_{rdma} \approx o + L + (m-1).G \tag{7}$$

$$T_{fallback} \approx o + L + o + (m-1).G \tag{8}$$

An additional o is observed for fall-back protocol, since the remote process/thread is involved in response to the get request. For bulk transfers, T_{rdma} and $T_{fallback}$ are $\approx m.G$, if the processes are tightly synchronized. However, $T_{fallback} \in \Omega(T_{rdma})$: Fall-back protocol requires an involvement of a remote process/thread to make progress on data transfer. T_{rdma} does not require an involvement from a remote process/thread for communication progress. Get is a dominant communication primitive in PGAS applications such as NWChem [11], hence using RDMA is critical for scaling these applications [16], [18].

The model for put communication is similar to the get model. The put primitive does not require a fall-back protocol because the put primitive follows similar buffer reuse semantics as MPI [3], [4]. PAMI's default RMA messaging primitive may be used when either memory region is not available.

The space complexity of communication protocols for contiguous data-type is independent of m, and partially dependent on the degree of non-blocking data transfers. This makes RDMA based protocols conducive for large scale systems such as Blue Gene/Q.

2) Uniformly Non-contiguous (Strided) Datatype Get/Put Communication: Uniformly non-contiguous (strided) data type is used in patch-based data transfer of distributed data structures. ARMCI provides interfaces to represent multi-dimensional patch requests with very little memory usage in comparison to the general I/O vector requests [4]. Figure 2 shows an example of strided put communication from process P_i's local buffer to patches (dashed lines) of processes P_r, P_s, P_t, and P_u, respectively.

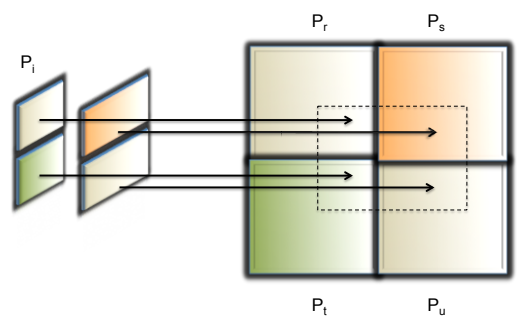

Figure 2. Example of Strided Communication, Process P_i writes rectangular Patches to Processes P_r, P_s, P_t, and P_u respectively

Legacy communication protocols for strided data type use pack/unpack strategies, which require data packing at local process and data unpacking at the remote process. These protocols require intermediate buffering and flow control for completing the RMA operation.

Modern networks provide high messaging rate and network concurrency, obviating a need for a pack/unpack protocol. An important implication is that individual chunks may use RDMA for data transfer, eliminating a need of flow control and remote progress. A high throughput efficiency of networks such as Blue Gene/Q interconnection network supports this argument. The proposed communication protocols for these data type post a list of non-blocking RDMA requests, leveraging the network concurrency effectively. For a message size (m), divided in (s) dimensions as (m =

[2]We ignore the g parameter for simplicity, without loss of generality

$\prod_{i=0}^{s-1} l_i$), l_0 is the size of the contiguous chunk, the latency ($T_{strided}$) is given as:

$$T_{strided} \approx o.\prod_{i=1}^{s-1} l_i + m.G = o.m/l_0 + m.G \qquad (9)$$

As Equation 9 suggests, $T_{strided}$ is inversely proportional to the size of the contiguous chunk[3]. This may be a problem for tall-skinny communication. The proposed approach uses PAMI typed data type communication for such transfers For a majority of strided data transfers, zero copy based protocol is used. The space complexity is $\in \Theta(k.m/l_0)$, where k is the number of outstanding message requests from the user.

D. A Case for Asynchronous Progress Threads

A significant subset of communication primitives map directly to RDMA. Another subset of important operations are Atomic Memory Operations (AMOs) such as read-modify-write instructions for load balance counters and generic accumulate operations. Unlike Cray Gemini Interconnect [18], [17] and InfiniBand [20], PAMI on Blue Gene/Q does not provide hardware support for generic AMOs on the Network Interface Card (NIC)[4]. At the same time, non-RDMA variant of get operation is not truly one-sided: the source of the get needs to make an explicit call to the progress engine of PAMI. In our proposed implementation, non-RDMA variant of get is used when either of the memory regions are not available. Hence, it is critical to accelerate these operations, which would need explicit progress engine calls.

The SMT architecture of Blue Gene/Q comes to rescue to address the hardware limitations for AMOs. Each SMT thread may not be sufficient for an independent process, however it may be used to schedule an asynchronous thread for communication progress. The asynchronous thread is used to accelerate the AMOs and non-RDMA variant of get operations. Active message requests from other processes are handled by the asynchronous thread. The other SMT threads may be used to schedule computation.

An implementation artifact of using an asynchronous thread is guarding the PAMI progress engine with locks. The inherent competition of using locks may starve the asynchronous thread from servicing remote requests. Conversely, the main thread may not be able to make progress on local completions, while the asynchronous thread holds the lock.

In our proposed design, this limitation is addressed by using multiple communication contexts (ρ). This enhancement improves the progress schedule for each thread with a slight increment in space complexity ($\rho.\epsilon$). Each communication context can make independent progress and receives completions targeted for that particular context.

[3]The term o is observed on each communication list, hence the overhead is a multiple of the number of chunks

[4]Additional lower layer mechanisms for AMO with restrictions are available, but they are not openly available

E. Handling Conflicting Memory Accesses

ARMCI scalable protocols layer provides location consistency [14] by tracking the read (get) and writes (put, accumulate) to the communication clique (ζ). A naive algorithm maintains a communication status for a target (cs_{tgt}) by appropriately setting it as read/write as the data requests are generated. An outstanding write to a process must be *fenced* before any read requests are serviced from that process. This approach scales well in space complexity $\in \Theta(\zeta)$, but it suffers from false positives.

Using $dgemm$ as an example, it can be noted that A, B and C ($C = A \cdot B$) matrices would be distributed equally among processes for memory and load balancing. To overlap communication with computation, a process would request multiple non-blocking gets and perform asynchronous accumulates to the C matrix. Under the naive algorithm, each get would require that the accumulates have been *fenced* before gets are serviced. However, this is not necessary, since the read and write requests are from different distributed data structures. Several applications use distributed data structures as read-only or write-only (put/accumulate), for which this situation is encountered.

The problem may be alleviated largely by locally setting communication status for each memory region (cs_{mr}) appropriately as the read/write(s) are requested. In our proposed approach, we use an 8-bit integer to set the cs_{mr} for each distributed data structure. Accumulate operations are associative - ordering among updates is not required. The space complexity is $\in \Theta(\sigma.\zeta)$.

IV. Performance Evaluation

This section presents a performance evaluation of our proposed design. A set of detailed preliminary communication benchmarks are used to understand the performance of different communication primitives in multiple scenarios. The section concludes with a detailed performance evaluation using NWChem [11], a petascale computational chemistry application designed using Global Arrays [5] and ARMCI [12]. The performance uses $ABCDET$ mapping for 5D torus. Up to 4096 processes (half-midplane) is used for performance evaluation (limited by the time allocation).

A. Fundamentals

Table II contains the attribute values used in performance evaluation. Values of α, β, γ, δ and ϵ are computed by calculating the actual time during program execution. The maximum data transfer size is 1 MB, which is covers a large percentile of message size used in real applications.

The endpoint space and time complexity is 4 bytes and 0.3 μs, which makes the memory consumption highly scalable for building even larger systems. The memory region space utilization is 8 bytes, which makes it very scalable caching active global address space structures (σ: 1-7) and local buffers (τ: 1-3). The communication clique (ζ) varies

	Property	Symbol	Value
1	Message Size for Data Transfer	m	16 Bytes - 1 MBytes
2	Total number of processes	p	2 - 4096
3	Number of processes/Node	c	1 - 16
4	Endpoint Space Utilization	α	4 Bytes
5	Endpoint Creation Time	β	.3 μs
6	Memory Region Space Utilization	γ	8 Bytes
7	Memory Region Creation Time	δ	43 μs
8	Context Space Utilization	ϵ	varies
9	Context Creation Time	ϱ	3821 - 4271 μs
10	Number of contexts	ρ	1 - 2
11	Communication Clique	ζ	1 - p
12	Number of Active Global Address Structure	σ	1 - 7
13	Number of Local Buffers used for Communication	τ	1 - 3

Table II
EMPIRICAL VALUES OF TIME AND SPACE ATTRIBUTES

from 1-p. ζ is not necessarily generated in an all-to-all communication step, but $gets$, and $accumulates$ to many processes during an application lifetime. The observation is relevant for NWChem [11], which uses load balance counters, and asynchronous $gets$ and $accumulates$ with little to no regularity in communication patterns.

B. Preliminary Empirical Analysis

1) Contiguous Datatype Performance: Figure 3 shows the raw latency performance for inter-node communication using $gets$ and $puts$. A get latency of 2.89 μs is observed for 16 bytes. RDMA is used in both primitives, as it is the preferred method for mapping get/put requests. The put latency is 2.7 μs, but it primarily indicates send overhead and local completion. This communication type reflects usage pattern for many kernels, which do not require the result of write (put or $accumulate$) to be available immediately. A drop at 256 bytes is observed in latency. This is primarily due to fact that cache aligned data transfer(s) are faster than unaligned data transfer ($m <$ 256 bytes).

Figure 4 shows the performance of put and get bandwidth using two processes and inter-node communication. The get overhead (due to round-trip) is visible till 8K bytes. The curves achieve a peak bandwidth of 1775 MB/s. Blue Gene/Q link bandwidth is 2 GB/s, with overhead a maximum of 1.8 GB/s is available for performance [1], [2]. A peak bandwidth of 1775 MB/s asserts the efficacy of the proposed design for raw communication benchmarks achieving $\approx$ 99% efficiency.

Figure 5 shows the effective latency/byte. The test is used to determine the inflection point which may be used for message aggregation by an application. This is useful for applications, which may send out many small messages over a period of time. Beyond 4K bytes, the latency/byte is $\approx$ 1ns. Figure 6 shows bandwidth efficiency: ratio of message bandwidth to peak bandwidth (1.8 GB/s). Blue Gene/Q torus interconnection network is highly effective for small/medium messages, $N_{1/2}$ (message size at which half of peak bandwidth can be achieved) is 2 Kbytes. An efficiency $\geq$ 90% is achieved beyond 16 Kbytes message size.

Figure 7 shows the get latency as a function of process rank. A pseudo-oscillatory curve is observed with cluster(s) of processes observing similar latency (due to same network distance from process 0). This is an artifact of using 5D torus - with $ABCDET$ mapping, the distance between process 0 and other process oscillates explaining the latency curve. Minimum observed latency is 2.89μs and the maximum latency is 3.38μs. With 2048 processes, 128 nodes are used (128 $\cdot$ 16 = 2048). Hence:

$$128 = 2(A) \cdot 2(B) \cdot 4(C) \cdot 4(D) \cdot 2(E) \tag{10}$$

where $n(dim)$: n is the size of the dimension and dim is the dimension name. With wrap-around, a maximum distance of $(2{+}2{+}4{+}4{+}2)/2 = 7$ is present. The difference between maximum and minimum latency is 0.49μs, hence each hop adds $0.49/(7 \cdot 2(round\text{-}trip))$ resulting of 35 ns latency increment per hop. This matches closely with the empirical values stated by Chen *et al.* [2]. Hence, the proposed approach does not incur any extra overhead in get communication at scale.

2) Strided Datatype Communication Performance: Figure 8 shows the performance of strided communication using get and put primitives with 1M Bytes data transfer. The curve is relatively identical to Figure 4 as l_0 (size of the contiguous chunk) is increased. It is a worthwhile observation that for each instance of $l_0 \leq$ 1M Bytes, multiple outstanding messages are initiated from the source.

3) Atomic Memory Operations (AMOs): **Load Balance Counters:** Load balance counters are an essential primitive for applications such as NWChem [11]. Figure 9 compares the average latency for read-modify-write (fetch-and-add) observed by processes $1 - p$ as fetch-and-add is requested on a counter resident at process 0. This is a micro-kernel of computation phases in NWChem [11] The figure shows performance with/without using asynchronous threads and with/without computation by process 0. It is observable that by adding computation to process 0 increases the average latency observed by each process significantly ($t_{compute}$ by process 0 is $\approx$ 300μs). The default and asynchronous thread

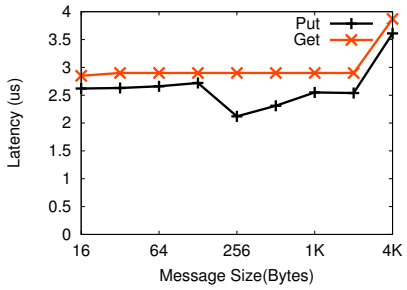

Figure 3. Raw Latency Performance

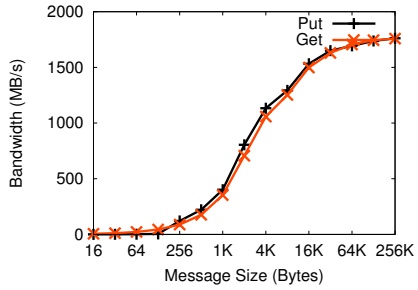

Figure 4. Raw Bandwidth Performance

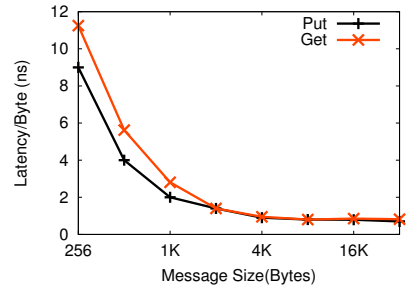

Figure 5. Effective Latency/Byte

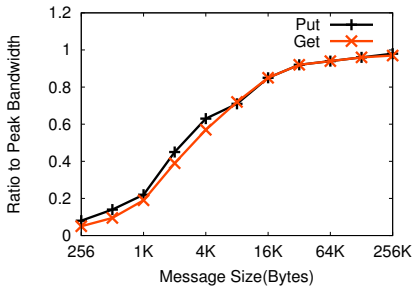

Figure 6. Bandwidth Efficiency

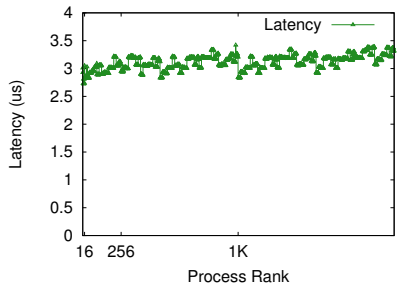

Figure 7. Get Latency Distance

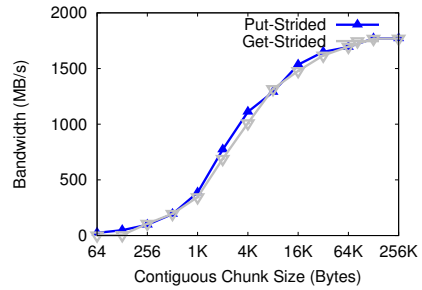

Figure 8. Strided Communication Performance as a function of l_0, m = 1 MBytes

based approach perform comparable to each other when process 0 is not involved in computation.

There are multiple inferences from these observations. Asynchronous threads are important for accelerating load balance counters. Without asynchronous thread based design, the time taken is dependent up on the explicit progress made by process 0.

However, even with asynchronous thread(s), the latency increases linearly with system size. This observation is in contrast with Cray Gemini [17], [18], [21], where a sub-linear (step) function is observed in performance. Given the high use of load balance counters, hardware assisted fetch-and-add such as ones with InfiniBand [20] and Cray Gemini can help accelerate the load balance counters without a need of asynchronous threads.

C. Performance Evaluation with NWChem: Computational Chemistry at Scale

1) Overview of NWChem Self Consistent Field Calculation: The NWChem Self Consistent Field (SCF) calculation involves a number of different compute phases. The first step is a calculation of molecular electron density matrix which is constructed from a sum of atomic contributions. The computation involves contracting the electron density matrix with 2-electron integrals that represent the electron-electron interaction to produce contributions to the Fock-matrix. Figure 10 shows the algorithm for SCF calculation. The local compute is abstracted as $do\ work$. The construction of fock matrix is accelerated using load balance counters. The processes request a fetch-and-add of the load balance (SharedCounter in Figure 10) counter, perform $gets$, compute locally and update the output matrix, as shown in Figure 10.

2) Performance Evaluation: The performance evaluation of SCF uses 6 molecules. This is smaller version of the original input deck (24 water molecules) to the Gordon Bell finalist used by Apra *et al.* for executing on full ORNL Jaguar system [22]. Figure 11 shows the performance using 6 water molecules on 1024, 2048 and 4096 processes, respectively. A total of 644 basis functions are used for the calculation. Performance is compared using default (D) and Asynchronous Thread (AT) based approach.

The asynchronous thread based approach outperforms the default approach by reducing the execution time up to 30%; the overall time in load balance counters reduces sharply for asynchronous thread based approach. This is explained as follows: For most of the calculation, the load balance counter is held by process 0. In the default (D) approach, any request to fetch-and-add on the counter is performed by the main thread, which is possible only when the main thread calls the PAMI progress engine. The delay observed by other processes is proportional to time process 0 spends in computation. The problem worsens further because as soon as process 0 gets another task to compute, other processes will have to wait for process 0 to finish to get the next task. With the AT approach, process 0 does not need to make explicit progress to perform fetch and add operation.

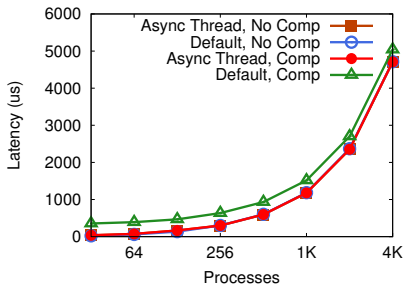

Figure 9. Rmw (Fetch-and-Add) Performance

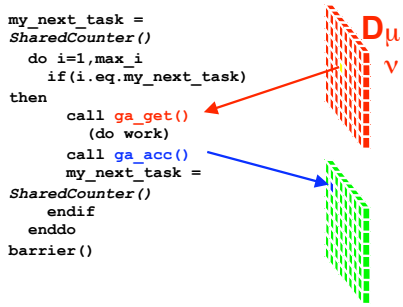

Figure 10. Self Consistent Field Algorithm

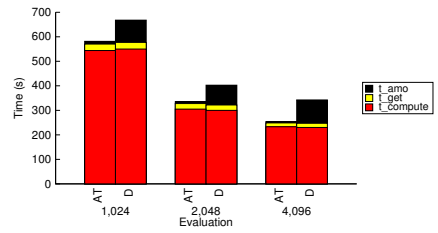

Figure 11. NWChem SCF, 6 Water Molecules; AT: Asynchronous Thread, D: Default implementation

Hence, it is absolutely critical to use asynchronous threads for accelerating the AMOs and other non-RDMA operations such as PAMI_Get.

V. Related Work

Designing scalable communication subsystems has been of interest to many research groups with primary focus on two-sided communication with MPI [3], [4].

Kumar *et al.* have provided an excellent description of PAMI on Blue Gene/Q, and some preliminary results [13]. However, they do not provide guidance on the applicability of PAMI primitives for PGAS communication subsystems such as ARMCI. GASNet [23] also provides PAMI conduit, however, we are not aware of any published research with GASNet on Blue Gene/Q. Additionally, GASNet primarily uses active messages, while the primary focus of ARMCI is supporting the RMA model. Many studies have focused on micro-architecture aspects of the Blue Gene/Q chip [1], [2].

Studies focusing on communication subsystem aspects of other interconnects is presented here: Multiple studies have been undertaken on designing scalable communication subsystems on Cray Gemini Interconnect [21], [18]. Scalable message passing for Cray Gemini Interconnect has been proposed [24]. A similar study for Charm++ has been undertaken [25]. However, these studies have focused on message passing using uGNI, while the focus of the paper is one-sided communication.

Scalable MPI design on InfiniBand has been addressed by OpenMPI and MVAPICH/MVAPICH2 with salient features such as RDMA, Shared Receive Queue (SRQ), Xtended Reliable Connection, Fault tolerance with Automatic Path Migration (APM) and multi-rail systems [26], [27]. Efforts for scalable MPI design in other Interconnects such as Quadrics, Myrinet, High Performance Switch have also been performed [28], [29]. However, none of the efforts above address the one-sided communication aspects, which are addressed in this article.

VI. Conclusions

This paper has presented a design of scalable PGAS communication subsystems on recently proposed Blue Gene/Q architecture using PAMI communication interface. The proposed communication infrastructure is used to design time-space efficient communication protocols for frequently used data-types (contiguous, uniformly non-contiguous) using Remote Direct Memory Access (RDMA) get/put primitives. We have accelerated the performance of load balance counters by using asynchronous threads. The synchronization traffic is reduced by tracking conflicting memory accesses in distributed space with slightly increment in space complexity. An evaluation with simple communication benchmarks show a get latency of 2.89μs and peak bandwidth of 1775 MB/s resulting in $\approx$ 99% communication efficiency. Accelerated load balance counters improve the execution time by up to 30% for NWChem self consistent field calculation.

VII. Acknowledgements

We would like to acknowledge Extreme Scale Computing Initiative at Pacific Northwest National Laboratory for funding this research. We would like to thank the reviewers for their comments and suggestions towards strengthening the paper. We would also like to thank Dr. Amith Mamidala, IBM and Dr. Jeff Hammond, ANL for suggestions on the design and implementation of ARMCI on IBM PAMI.

References

[1] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, G.-T. Chiu, P. Boyle, N. Chist, and C. Kim, "The ibm blue gene/q compute chip," *Micro, IEEE*, vol. 32, no. 2, pp. 48 –60, march-april 2012.

[2] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The ibm blue gene/q interconnection fabric," *Micro, IEEE*, vol. 32, no. 1, pp. 32 –43, jan.-feb. 2012.

[3] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, 1996.

[4] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. L. Lusk, W. Saphir, T. Skjellum, and M. Snir, "MPI-2: Extending the message-passing interface," in *Euro-Par, Vol. I*, 1996, pp. 128–135.

[5] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global Arrays: A Nonuniform Memory Access Programming Model for High-Performance Computers," *Journal of Supercomputing*, vol. 10, no. 2, pp. 169–189, 1996.

[6] P. Husbands, C. Iancu, and K. A. Yelick, "A Performance Analysis of the Berkeley UPC Compiler," in *International Conference on Supercomputing*, 2003, pp. 63–73.

[7] R. W. Numrich and J. Reid, "Co-array fortran for parallel programming," *SIGPLAN Fortran Forum*, vol. 17, pp. 1–31, August 1998.

[8] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing," in *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2005, pp. 519–538.

[9] B. Chamberlain, D. Callahan, and H. Zima, "Parallel Programmability and the Chapel Language," *International Journal on High Performance Computing Applications*, vol. 21, no. 3, pp. 291–312, 2007.

[10] A. Vishnu, H. Van Dam, W. De Jong, P. Balaji, and S. Song, "Fault Tolerant Communication Runtime Support for Data Centric Programming Models," in *International Conference on High Performance Computing*, 2010.

[11] R. A. Kendall, E. Aprà, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus, and A. T. Wong, "High Performance Computational Chemistry: An Overview of NWChem, A Distributed Parallel Application," *Computer Physics Communications*, vol. 128, no. 1-2, pp. 260–283, June 2000.

[12] J. Nieplocha and B. Carpenter, "ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems," in *Lecture Notes in Computer Science*. Springer-Verlag, 1999, pp. 533–546.

[13] S. Kumar, A. R. Mamidala, D. Faraj, B. E. Smith, M. Blocksome, B. Cernohous, D. Miller, J. Parker, J. Ratterman, P. Heidelberger, D. Chen, and B. D. Steinmacher-Burow, "Pami: A parallel active message interface for the blue gene/q supercomputer," in *International Parallel and Distributed Processing Symposium*, 2012, pp. 763–773.

[14] G. R. Gao and V. Sarkar, "Location consistency-a new memory model and cache consistency protocol," *IEEE Transactions on Computers*, vol. 49, pp. 798–813, 2000.

[15] Subsurface Transport over Multiple Phases, "STOMP," http://stomp.pnl.gov/.

[16] A. Vishnu and M. Krishnan, "Efficient On-demand Connection Management Protocols with PGAS Models over InfiniBand," in *International Conference on Cluster, Cloud and Grid Computing*, 2010.

[17] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, aug. 2010, pp. 83 –87.

[18] A. Vishnu, M. ten Bruggencate, and R. Olson, "Evaluating the potential of cray gemini interconnect for pgas communication runtime systems," in *Proceedings of the 2011 IEEE 19th Annual Symposium on High Performance Interconnects*, 2011, pp. 70–77.

[19] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: Incorporating long messages into the logp model — one step closer towards a realistic model for parallel computation," Santa Barbara, CA, USA, Tech. Rep., 1995.

[20] InfiniBand Trade Association, "InfiniBand Architecture Specification, Release 1.2," October 2004.

[21] A. Vishnu, J. Daily, and B. Palmer, "Scalable pgas communication subsystem on cray gemini interconnect," in *Interational Conference on High Performance Computing*, Pune, India, 2012.

[22] E. Aprà, A. P. Rendell, R. J. Harrison, V. Tipparaju, W. A. deJong, and S. S. Xantheas, "Liquid Water: Obtaining The Right Answer For The Right Reasons," in *SuperComputing*, 2009.

[23] D. Bonachea, "GASNet Specification, v1.1," October 2002.

[24] H. Pritchard, I. Gorodetsky, and D. Buntinas, "A uGNI-based MPICH2 nemesis network module for the cray XE," in *Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface*, ser. EuroMPI'11, 2011, pp. 110–119.

[25] Y. Sun, G. Zheng, L. V. Kale, T. R. Jones, and R. Olson, "A uGNI-based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect," in *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Shanghai, China, May 2012.

[26] J. Liu, A. Vishnu, and D. K. Panda, "Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation," in *SuperComputing*, 2004, pp. 33–44.

[27] A. Vishnu, M. J. Koop, A. Moody, A. R. Mamidala, S. Narravula, and D. K. Panda, "Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective," in *Cluster Computing and Grid*, 2007, pp. 479–486.

[28] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, pp. 46–57, 2002.

[29] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su, "Myrinet: A Gigabit-per-second Local Area Network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, February 1995.