Designing Scalable PGAS Communication Subsystems on Cray Gemini Interconnect

Abhinav Vishnu [#1] , Jeff Daily [#2] , and Bruce Palmer [#3]

[#] *Pacific Northwest National Laboratory*
902 Battelle Blvd, Richland, WA 99352
[1] abhinav.vishnu@pnnl.gov
[2] jeff.daily@pnnl.gov
[3] bruce.palmer@pnnl.gov

***Abstract*—The Cray Gemini Interconnect has been recently introduced as a next generation network architecture for building multi-petaflop supercomputers. Cray XE6 systems including LANL Cielo, NERSC Hopper, and the proposed NCSA Blue-Waters, as well as the Cray XK6 ORNL Titan leverage the Gemini Interconnect as their primary Interconnection network. At the same time, programming models such as the Message Passing Interface (MPI) and Partitioned Global Address Space (PGAS) models such as Unified Parallel C (UPC) and Co-Array Fortran (CAF) have become available on these systems. Global Arrays is a popular PGAS model used in a variety of application domains including hydrodynamics, chemistry and visualization. Global Arrays uses Aggregate Remote Memory Copy Interface (ARMCI) as the communication runtime system for Remote Memory Access (RMA) communication. This paper presents a design, implementation and performance evaluation of scalable and high performance communication ARMCI on Cray Gemini. The design space is explored and time-space complexities of communication protocols for one-sided communication primitives such as contiguous and uniformly non-contiguous datatypes, atomic memory operations (AMOs) and memory synchronization is presented. An implementation of the proposed design (referred as ARMCI-Gemini) demonstrates the efficacy on communication primitives, application kernels such as LU decomposition and applications such as Smooth Particle Hydrodynamics (SPH).**

I. Introduction

Scalable programming models are pivotal in extracting the peak performance of of high-end systems. While the Message Passing Interface (MPI) [1], [2] is ubiquitous, Partitioned Global Address Space (PGAS) programming models are gaining traction with performance portability and productivity. PGAS models rely on underlying communication subsystems to abstract network interfaces and provide one-sided communication primitives for get/put, atomic memory operations, synchronization and datatypes. Scalable and convenient abstractions have resulted in an increased adoption of PGAS models [3].

Global Arrays [4] (GA) is a popular PGAS model used in science domains such as computational fluid dynamics, groundwater simulation, astrophysics, climate modeling, and chemistry [5], [6]. Global Arrays uses the Aggregate Remote Memory Communication Interface (ARMCI) [7] for network communication. The scalability of GA and its related applications depend on time-space efficient protocols of the ARMCI communication runtime. To meet these objectives, design of scalable communication protocols on native high-speed interconnects has been proposed on today's advanced systems such as InfiniBand [8], Cray XTs, and IBM BlueGene systems [9], [10].

Recently, the Cray Gemini Interconnect has been introduced as a next generation network architecture for building multi-petaflop supercomputers. The Cray Gemini Interconnect is integrated as a part of Cray XE6/XK6 systems. The prominent features include hardware support for remote memory access (RDMA), atomic memory operations (AMOs) and symmetric heap to accelerate the transition of communication libraries to the newly proposed architecture. Cray XE6 systems including LANL Cielo, NERSC Hopper, and the proposed NCSA Blue-Waters, as well as the Cray XK6 ORNL Titan use the Gemini Interconnect as their primary interconnection network. Applications which use GA such as Smooth Particle Hydrodynamic (SPH), NWChem must extract the best performance on these systems to advance the science. Hence, it is critical to design a scalable PGAS communication subsystem on Cray Gemini Interconnect.

A. Contributions

The major contributions of the paper are:

- A detailed discussion on Cray Gemini userspace libraries and applicability of these libraries for PGAS models. The paper presents a performance and productivity argument in using the userspace libraries.
- An exploration of solution space using Distributed Shared Memory Application (DMAPP) [11] interface and propose a native one-sided communication runtime system using ARMCI (The proposed design is referred as ARMCI-Gemini). The communication protocols which utilize the network concurrency and communication/computation overlap protocol for accumulate operations are presented. The time-space complexity of communication protocols for contiguous, uniformly non-contiguous datatype communication, atomic memory operations and memory synchronization is analyzed.
- The proposed design (ARMCI-Gemini) is implemented and the performance is evaluated using up to 8192 processes. The efficacy of ARMCI-Gemini is demonstrated using communication primitives (contiguous, non-

978-1-4673-2371-0/12/$31.00 ©2012 IEEE

contiguous, atomic memory operations), communication benchmarks (shift, transpose), application kernels (LU decomposition, Lennard Jones simulation) and an application (Smooth Particle Hydrodynamics). ARMCI-Gemini achieves a get latency of 1.3 μs and a peak bandwidth of 6 GB/s, incurring negligible overhead on DMAPP userspace library.

The rest of the paper is organized as follows: Section II provides a brief background of the proposed work. Section III presents possible solutions for designing scalable and high performance PGAS communication runtime (referred as ARMCI-Gemini) on the Cray Gemini Interconnect. Section IV presents a performance evaluation of ARMCI-Gemini. Section V provides related work on scalable communication runtime systems for high performance interconnects. Finally, Section VI presents some conclusions and plans for future work.

II. Background

This section presents a background of the proposed work. A description of the Cray Gemini Interconnect [11] is followed by a brief description of ARMCI [7].

A. Cray Gemini Interconnect

Figure 1 shows a block diagram of the Cray Gemini ASIC. Each Gemini ASIC has two Network Interface Cards (NICs), and a 48-port Cray YARC2 router. The NICs within an ASIC (Inter-ASIC) are connected to a Netlink block, which provides internal routing between the NICs. A total of eight links connect a Netlink block to a router. Each Gemini ASIC is connected to other ASICs using a 3D Torus topology. A total of eight links each are used to connect in the 'x' and 'z' dimensions, and four links are used in the 'y' dimension [13], [11] (The other four links in the 'y' dimension are used by the Netlink block).

Multiple routing algorithms are supported on the Cray Gemini Interconnect: deterministic, hashed and adaptive. "The deterministic routing algorithm follows dimension-order routing of messages with predetermined links within each dimension. The hashed dimension-order routing algorithm provides more flexibility in selecting a link on various dimensions. The adaptive dimension-order routing allows the packets to be scheduled on lightly loaded links adaptively" [13], [11].

The Cray Gemini Interconnect supports Programmed I/O using the Fast Memory Access (FMA) protocol, which is used up to 4096 bytes in the proposed design. Larger message sizes use Block Transfer Engine (BTE) based protocols, which use doorbell mechanism for Remote Direct Memory Access (RDMA).

B. Aggregate Remote Memory Copy Interface

"ARMCI [7] is a communication runtime system which provides general-purpose, efficient, and widely portable remote memory access (RMA) operations (one-sided communication) optimized for contiguous and non-contiguous (strided, scatter/gather, I/O vector) data transfers. ARMCI provides interfaces for collective/non-collective memory allocation, atomic memory operations (fetch-and-add, compare-and-swap), bulk accumulate operations and mutex operations." [14], [11]

ARMCI supports the location consistency model. A set of ARMCI interfaces support pairwise memory synchronization and collective memory synchronization. ARMCI supports non-blocking interfaces with explicit and implicit handles and adheres to buffer reuse semantics similar to MPI [1], [2]. ARMCI is the underlying communication subsystem for higher level programming models such as Global Arrays [4] and Co-Array Fortran [15]. The Global Arrays programming model provides abstractions for distributed arrays and leverages the communication primitives provided by ARMCI. Figure 2 shows the software ecosystem of Global Arrays and ARMCI.

ARMCI leverages the low level network primitives provided by modern networks. It is supported on clusters with commodity interconnects (InfiniBand, Ethernet) and high-end systems (IBM Blue Gene/L, Blue Gene/P, and Cray XT5/XT4). The proposed implementation in this article will be available in future releases.

III. ARMCI-Gemini Design

This section presents solution space for designing scalable and high performance PGAS communication runtime systems on the Cray Gemini Interconnect. This involves designing efficient time-space complexity protocols leveraging the properties of the interconnect. A detailed discussion on the design choices - symmetric heap allocation, network concurrency for non-contiguous datatype communication, hardware mechanisms for load balance counters, adaptive protocols for accumulate operations and constant time memory synchronization is presented.

A. Userspace Libraries

The Cray Gemini Interconnect supports two userspace libraries - uGNI and DMAPP. These userspace libraries are specifically designed for Message Passing (MPI) and PGAS communication runtime systems, respectively. Table I shows the primary differences between the properties of these userspace libraries.

	Property	DMAPP	uGNI
1	EndPoint Management	No	Yes
2	Registration Cache	No	Yes
3	Remote Completion Notification	No	Yes
4	Symmetric Heap Allocation	Yes	No

TABLE I
PROPERTIES OF USERSPACE LIBRARIES SUPPORTED BY CRAY GEMINI INTERCONNECT

The DMAPP library abstracts endpoint management and the caching of memory registration for local communication buffers. DMAPP uses the $udreg$ module for maintaining local registration cache entries which are shared with other user libraries such as the MPI runtime. An advantage of the sharing is interoperability with the MPI runtime without duplicating registration cache entries. Automatic endpoint management is

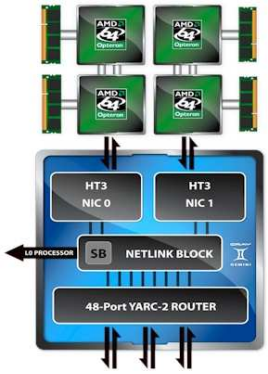

Fig. 1. A Block Diagram of Cray Gemini ASIC [12]

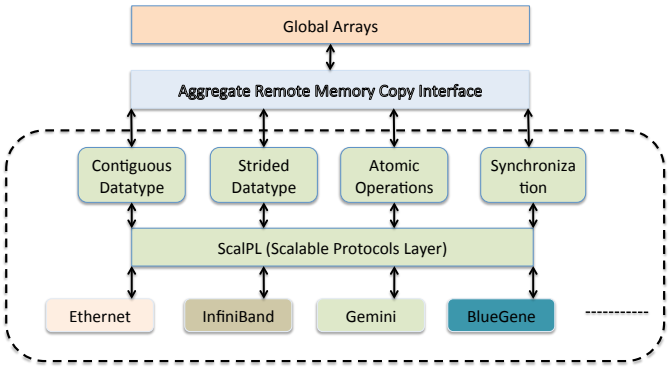

Fig. 2. Global Arrays Software Architecture (Designed Components are Shown in Green)

a useful abstraction which reduces the effort of designing a PGAS communication runtime system.

The DMAPP library supports symmetric heap allocation. The symmetric heap allocation functionality is space-time efficient. It eliminates a need for collective address exchange of each memory allocation. For p processes which exchange a buffer of size m (size of base address pointer of each global address allocation) using all-to-all broadcast (equivalent of MPI_Allgather), the overall startup time reduces by $\Theta(p{\cdot}m)$ time units for each global address space allocation.

A high performance PGAS communication subsystem would cache base addresses of a large subset of processes requiring $O(p{\cdot}m)$ units of address space. Some implementations may reduce the space and time complexity by using dynamic caching of base address pointers and eviction strategies by requiring address lookup in the critical path. The symmetric memory allocation eliminates a need for address exchange completely. The primary limitation of symmetric heap allocation is that the size of the symmetric heap needs to be predefined before application startup. All processes need to use a similar amount of memory during allocation to eliminate address exchange on the fly.

The symmetric heap allocation is not useful for local communication buffers (equivalent of ARMCI_Malloc_local). The size of the local communication buffers may vary between processes, which would result in different offsets, if this approach was to be used. This does not result in additional complexity, since the DMAPP layer abstracts buffer registration completely.

B. Elements of PGAS Communication Runtime

This section presents design choices for a scalable PGAS communication runtime system on the Cray Gemini Interconnect. The section is divided into protocols for contiguous datatype, uniformly non-contiguous datatype, atomic memory operations and synchronization.

1) Contiguous Datatype: Contiguous datatype transfer is fundamental to many data transfer requests in PGAS models. Remote Direct Memory Access (RDMA) semantics provided by commodity and proprietary interconnects such as InfiniBand [8], and the Cray Gemini Interconnect [11] match very well with the contiguous datatype transfer. These networks require that the source and target communication buffers are contiguous. The buffers are required to be explicitly registered to prevent the kernel from swapping them out during communication.

The DMAPP library abstracts the requirement of registration entirely from the user by using an internal registration cache for local buffers and symmetric heap allocation for global address space data. Hence, the contiguous datatype transfer incurs negligible overhead on the native RDMA performance.

Let p represent the number of processes, k represent the maximum number of implicit outstanding data transfer requests from a source (runtime parameter), and d represent the size of a communication handle. In the proposed design, the memory requirement for contiguous data transfer is $\Theta(k{\cdot}d)$. In Cray Gemini interconnect, d is in the order of bytes. For scalable user-level algorithms, it is expected that the number of outstanding data transfers (from one source to all targets) to be $O(\log(p))$ or $O(\sqrt{p})$. For the proposed implementation, a linear model is used for the number of outstanding messages. The y-intercept is a predefined constant c and the slope as $\log(p)$, resulting in memory utilization to be $\Theta(d.(c + \log(p)))$.

2) Non-contiguous Datatype: Non-contiguous datatype transfer is frequently observed with many scientific applications. Many scientific applications read/update a patch of a distributed data structure (such as a distributed arrays in Global Arrays [4]), which results in a non-contiguous datatype transfer. Application kernels with temporal communication properties (such as multi-point stencil) update data on multiple processes in the grid. Updates are contiguous in some dimensions and non-contiguous in other dimensions.

A special type of non-contiguous datatype communication is strided (uniformly non-contiguous) datatype communication, which results in a read/update of a rectangular patch of global address space data. These datatypes can be expressed using the datatype interface in MPI [1], [2]. Communication runtime systems such as ARMCI [7] provide explicit interfaces

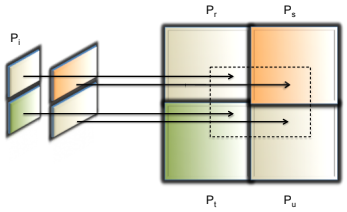

Fig. 3. Example of Strided Communication

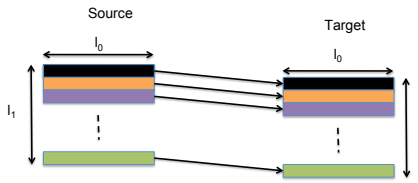

Fig. 4. Pipelined Communication Protocol for Strided Put Communication Primitive

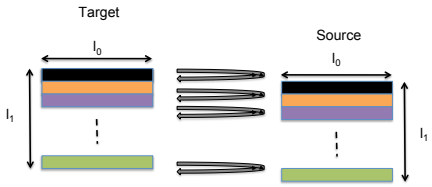

Fig. 5. Pipelined Communication Protocol for Strided Get Communication Primitive

for strided communication, due to their frequent usage by applications. Memory requirements to express strided communication is $\Theta(n)$, where n is the number of dimensions of data distribution. The frequent usage of strided communication makes it necessary to design scalable communication protocols for these datatypes.

Figure 3 shows an example of strided put communication in ARMCI [7]. In the example, a process P_i updates a patch of global address space data by expressing the dimensions of the patch. The underlying metadata manager queries the data distribution and transforms the request into multiple strided datatype put communications from the nodes which hold the data. Each data request from process P_i to P_r, P_s, P_t, and P_u results in a strided datatype communication.

Unlike contiguous datatype communication, strided datatype communication is not supported natively in the Cray Gemini hardware. The *Send-Gather, Receive-Scatter* functionality available with InfiniBand [8] is not truly one-sided. The solution space for communication protocols involves eager and rendezvous protocols for communication similar to MPI communication protocols.

The eager protocol involved local packing of the data; sending the data to an intermediate buffer on the remote node and unpacking of the data on the remote node. To expedite communication progress, asynchronous agent(s) are used. The previous approaches to designing scalable PGAS communication subsystems involve using one or more asynchronous agent(s) [16], [17]. Copy based protocols are accelerated using memory allocation which is shared among processes on a compute node. The agents are allowed to perform read/update on behalf of processes on the remote node. The copy based protocols require flow control, intermediate buffering and frequently suffer from overhead due to tradeoffs between polling and interrupt driven approaches.

Let M represent the size of an intermediate packing buffer, p represent the number of processes, β represent the overall size of data transfer. For $\beta \leq M$, the space requirement of the above design is $\Theta(M \log(p))$ on each process to allow a $\log(p)$ number of outstanding messages at any point . Multiple intermediate packing buffers may be required when $\beta > M$. The space requirement at the asynchronous agent is $(M \cdot p)$ to support eager protocol in the worst case. The rendezvous protocol may use a buffer pool to reduce the space complexity for intermediate buffers on the processes and asynchronous agent(s). However, this result cost of additional control messages in the rendezvous protocol.

To alleviate the memory cost of legacy protocols and effectively utilize the high network concurrency of modern interconnects, a zero copy based communication protocol is used for strided communication. The communication protocol performs a pipelined data transfer of individual contiguous chunks using non-blocking DMAPP communication primitives.

Figure 4 shows the communication protocol for strided put data transfer. Each contiguous chunk of data is transferred using the implicitly non-blocking data transfer DMAPP interface. When the total number of contiguous chunks exceed the maximum number of allowed non-blocking communication requests, the implementation waits for the completion of outstanding data transfers. The get strided communication primitive is implemented similarly using DMAPP non-blocking get communication primitive. The put/get primitives provide faster design and implementation of strided protocols on Cray Gemini Interconnect.

Let s represent the number of dimension(s) in strided datatype communication and l_i represent the number of bytes in i_{th} dimension. The space requirements of the pipelined communication protocol is $d.\prod_{i=1}^{s-1} l_i$ (Note that $\beta = \prod_{i=0}^{s-1} l_i$). The number of bytes in the first dimension is the contiguous chunk for the pipelined transfer.

There are multiple methods to scale this memory requirement. Most data transfers use up to four dimensions of read/update. The memory requirements of each transfer is proportional to the size of descriptor, which is on the order of bytes. In certain cases, when the total number of outstanding requests are greater than k, the progress engine is invoked to wait on the completion of some entries. The progress on these entries does not require an explicit communication progress from the remote process due to the use of RDMA. The number of outstanding data transfers is inversely proportional to l_0. The proposed solution is still very scalable since $d \ll M$.

3) Atomic Memory Operations: Atomic memory operations are critical communication primitives for PGAS models. They provide one-sided reduction of remote data. AMOs are used for load balance counters (atomic fetch-and-add), lock/unlock (atomic compare-and-swap) and accumulate (atomic add) operations. The load balance counters and lock/unlock operations operate on 64-bit value. The accumulate operation may be performed on bulk global address space data (matrix multiplication is a use case for bulk accumulates).

DMAPP provides quad-word based AMOs (atomic fetch-and-add, atomic compare-and-swap, atomic add, atomic and, atomic xor etc), which have native hardware support. An advantage of using hardware based AMOs is that an asynchronous agent is not required for performing AMOs. The AMOs on Cray Gemini Interconnect are fast due to caching of the AMO data on the NIC [13]. Hence, the associated cache line is resident on the NIC, until it is evicted. This results in a lower overhead in comparison to software based AMOs, particularly during the burst modes of load balancing and work stealing in applications such as Lennard Jones. The caching on the NIC does incur an overhead for local work stealing. DMAPP AMOs may not be used with CPU based AMOs on the same area of memory, since CPU based AMOs are not coherent with NIC based AMOs.

The accumulate operation maps well to DMAPP using the non-blocking atomic add operation. Each atomic add is a 64-bit operation. The number of data transfer requests is $\frac{m}{8}$, where m is the size of data transfer request. This is a reasonable protocol for small size accumulates. The number of 64-bit transactions increase with the size of the accumulate data.

For bulk accumulates, a $\{$lock $+$ pipelined *get-local accumulate-put* $+$ unlock$\}$ protocol is used. In this protocol, the key is to overlap the get and put phases of the protocol with the local accumulate. To achieve this overlap, multiple local buffers are used for requesting `NbPut` and `NbGet` phases of the update/read respectively. Figure 6 shows an example of the overlap protocol with triple-buffering. For triple buffering, the first wait on `NbPut` takes place in the third phase of the protocol. The pre-fetch phase of the protocol requests on `Get` to kick off the computation.

This protocol also requires the target of get to be an intermediate buffer. To achieve a high degree of overlap, v-buffering protocol may be used, each requiring communication buffers of size b (size is determined offline by performing bandwidth tests). The space complexity of this protocol is $\Theta(v \cdot b)$. The proposed implementation uses triple-buffering.

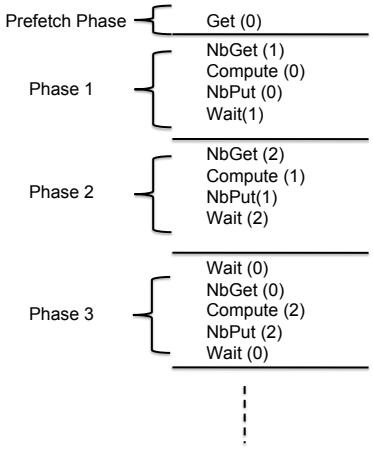

Fig. 6. Overlap Phase of get-local_acc-put Accumulate Protocol with triple-buffering Protocol

4) Synchronization: Scalable synchronization in PGAS models is critical since it involves memory and control synchronization. The control synchronization may be achieved by using an MPI barrier which has a time complexity of $\Theta(\log(p))$. Theorem III-B4 proves that the time complexity of memory synchronization for ARMCI-Gemini is $\Theta(1)$.

Theorem 1. *In the asynchronous agent based design, Let N denote the number of nodes, and ψ the number of asynchronous agents on the node. The cost of memory synchronization is $O(\psi.N)$*

Proof: In the asynchronous agent based design, the pack/unpack based communication protocols result in data transfer to intermediate buffers. The completion of the data transfer at initiator only indicates the data availability in the remote intermediate buffer, and not necessarily the user buffer. At the memory synchronization step, each process needs to send a *request to fence* message to each asynchronous agent involved in the pack/unpack based protocol and wait for *fence acknowledgment*, resulting in the cost of synchronization to be $O(\psi.N)$. ■

The $O(\psi.N)$ cost of memory synchronization tends to become a stricter bound with strong scaling. With increasing scale, the data is partitioned in more processes, resulting in communication with a larger number of processes, particularly for task-based models. This results in memory synchronization with a larger set of processes.

Corollary 1. *The cost of memory synchronization in ARMCI-Gemini is $\Theta(1)$.*

Proof: The proposed design leverages the location consistency semantics provided by the DMAPP library. Each communication primitive reads/updates user buffer(s) directly, precluding a need for *request to fence* and *fence acknowledgment* protocol. The memory synchronization in ARMCI-Gemini implementation is a *no-op*. ■

C. Implementation Details

The proposed protocols presented in the previous sections are implemented using ARMCI [7]. An advantage of selecting ARMCI is that it is used as a communication runtime for Global Arrays - a programming model used in many applications.

Registration/Deregistration of Local Buffers: DMAPP does not require explicit registration of local buffers. DMAPP maintains an internal registration cache. An important consequence of not registering local buffers is a possibility of many registration entries in the cache. This situation may be circumvented by explicitly registering the buffer immediately after allocation. Due to the high cost of deregistration on the fly, the local buffers are only deregistered during the finalization step of ARMCI.

Ordering Issues: We use adaptive routing with ARMCI-Gemini to alleviate congestion in the network. We also use relaxed ordering semantics for hypertransport for performance

issues. The current version of DMAPP supports location consistency with these parameters, which is required for ARMCI communication primitives.

Current Limitations: ARMCI-Gemini supports functionality limited to the Remote Memory Access communication model. ARMCI-Gemini does not support an Active Message communication model, partly due to missing functionality of Active Messages in the DMAPP interface. user-level Gemini Network Interface (uGNI) supports active messages. Another limitation of ARMCI-Gemini is the requirement of symmetric heap allocation. These limitations will be addressed in the future papers.

IV. Performance Evaluation

This section presents a performance evaluation of ARMCI-Gemini using communication benchmarks and application kernels. The objective of the performance evaluation is to provide guidance to application scientists and PGAS model designers on the expected performance from communication primitives and PGAS data distributions.

A. Experimental Testbed

The NERSC Hopper Phase II system [12] is used for performance evaluation. The Hopper Phase II system is a Cray XE6 system (code name Baker), which has 6384 nodes with two twelve-core AMD Magny-Cours 2.1 GHz processors on each node. Each node has 24 cores providing a total of 153,216 cores in the system. A total memory of 32 GB DDR3 1333 MHz is available per node. Each AMD core has an independent L1 and L2 cache of sizes 64KB and 512 KB, respectively. A 6MB L3 shared cache connects six magny-cours processors. The Hopper Phase II system provides a peak performance of 1.28 PF/s.

B. Performance Evaluation with Communication Benchmarks

This section presents a performance evaluation of ARMCI-Gemini using contiguous, strided datatype communication, and atomic memory operations. Each of these tests (except registration/deregistration, which requires only one process) uses two processes scheduled on two different nodes within a Cray Gemini ASIC.

1) Contiguous Datatype Performance: Figure 7 demonstrates the latency of the get communication primitive and compares the performance of ARMCI-Gemini implementation with a similar benchmark designed directly using the DMAPP interface [11]. A minimum message size of 16 bytes is used largely because many applications send a minimum of two doubles (16 bytes). The latency observed by the ARMCI-Gemini implementation is similar to latency of DMAPP benchmark. The primary reason is the absence of a protocol overhead in using ARMCI-Gemini over DMAPP. Both implementations use RDMA for contiguous data transfer, and overhead for searching the local registration cache (inside DMAPP library) and posting a descriptor is similar in both implementations. The observed latency for 16byte message is 1.3μs.

Figure 8 demonstrates the bandwidth observed for the get communication primitive using get communication benchmarks designed with ARMCI-Gemini and DMAPP implementations. It is observed that both implementations perform similarly. The peak bandwidth observed by these implementations is 6611 MB/s. The bandwidth test does not exhibit the concurrency of the Cray Gemini Interconnect because there is only one outstanding message between the communicating processes. This is particularly important for medium size messages, which typically do not achieve the peak bandwidth. The network concurrency is utilized with the strided communication benchmarks.

Figure 10 shows the latency observed for the put communication primitive using ARMCI-Gemini and DMAPP implementations. We observe that DMAPP and ARMCI-Gemini implementations achieve a latency of 1μs for 16byte messages. The ARMCI-Gemini implementation does not incur any overhead in communication protocol for contiguous data transfer. Figure 11 shows the bandwidth observed for the put communication primitive. The peak bandwidth achieved by DMAPP and ARMCI-Gemini implementations is 6600 MB/s. We observe a drop in bandwidth at 4Kbytes, which is due to a change from using the Fast Memory Access (FMA) based communication protocol to the Block Transfer Engine (BTE) protocol for both implementations [13], [18]. A discrepancy is observed with a bandwidth drop in DMAPP at 32Kbytes, which is likely due to system noise.

2) Strided Datatype Communication Performance: Figures 12 and 9 show the performance of strided put and strided get communication primitives, respectively. In each of these tests, a data transfer of M bytes is split in j chunks, each with contiguous size of s bytes ($M = j \cdot s$). The charts demonstrate bandwidth for different s sizes. These tests demonstrate the concurrency available with the Cray Gemini Interconnect. For a constant size M, if $s \approx M$, the peak bandwidth achieved is similar to the contiguous datatype. With increasing j, the network concurrency is the rate limiting factor resulting in lower bandwidth.

These tests are important in designing the data distribution (block, block-cyclic) and replication strategies (partial, complete) for distributed data structures. Application scientists may use these charts and determine the performance limiting factor in their algorithm using these performance results.

3) Atomic Memory Operations: **Load Balance Counters:** AMOs are frequently used for load balance counters and accumulate operations. A communication benchmark in which every process performs an `ARMCI_Rmw` (fetch-and-add) operation on memory exposed by process 0 is designed. This communication scenario is a micro-kernel of many science domains including Lennard Jones simulation, chemistry and bioinformatics in which each process requests a unit of work based on the value of load balance counter. The performance evaluation is shown in Figure 13.

It is observable that hardware assisted fetch-and-add scales very well with 5μs latency for 8192 processes, largely due to caching of atomics on the NIC. The graph demonstrates a

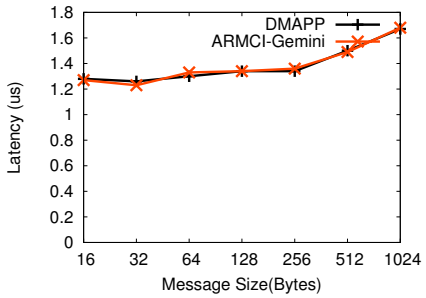

Fig. 7. Get Latency

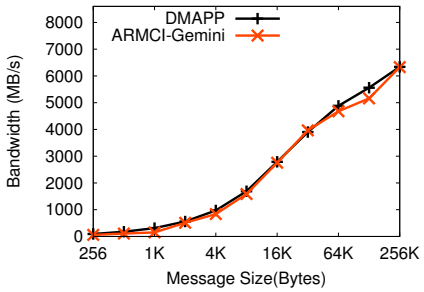

Fig. 8. Get Bandwidth

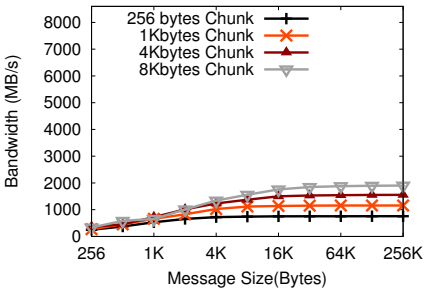

Fig. 9. Strided Get Bandwidth

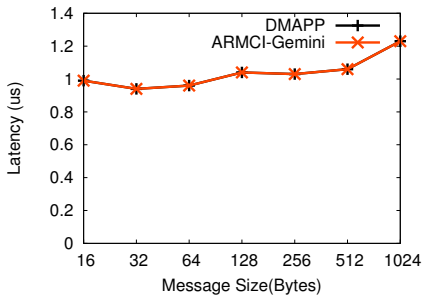

Fig. 10. Put Latency

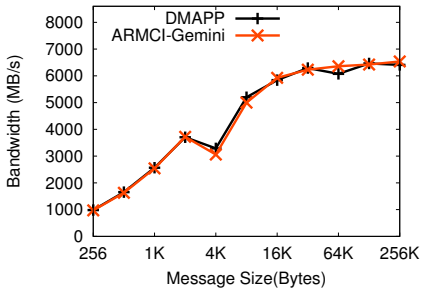

Fig. 11. Put Bandwidth

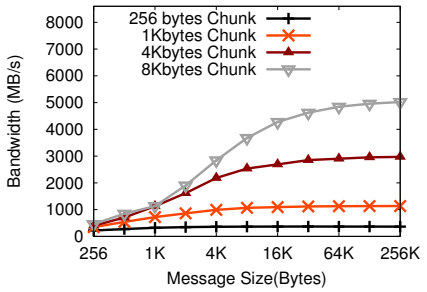

Fig. 12. Strided Put Bandwidth

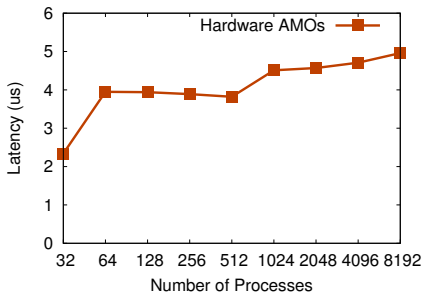

Fig. 13. Rmw (Fetch-and-Add) Latency

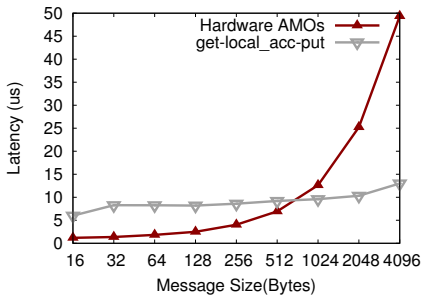

Fig. 14. Small Messages Accumulate Latency

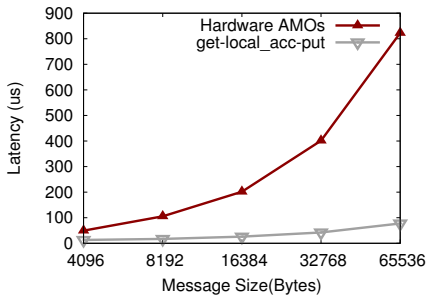

Fig. 15. Large Messages Accumulate Latency

step pattern in observed latency. The first step is observed at 64 processes - some (up to 48) of the processes are within the Cray Gemini ASIC, and other processes are outside the ASIC. The other step (1K processes) is observed due to increased hop counts of processes to perform atomic operation on process 0.

Accumulate Operations: Figure 14 demonstrates the performance of the accumulate operation for contiguous datatype comparing the performance of hardware-based AMOs and get-local_acc-put protocol. A crossover point at 512 bytes is observed, beyond which the get-local_acc-put based protocol outperforms hardware AMOs protocol. For messages smaller than 512 bytes, an extra overhead is observed during the lock and unlock phase of get-local_acc-put communication protocol. With increasing message size, the cost of lock/unlock operation is amortized by the cost of other phases of the protocol. The hardware AMOs based protocol needs to send out $\frac{m}{8}$ number of messages, which does not scale well with increasing message size. Figure 15 shows the performance of contiguous accumulate operation for large messages.

4) Shift and Transpose Communication Operations: The shift and transpose communication routines are used by Global Arrays functions such as `ga_transpose` and collective communication primitives. These communication operations demonstrate a time complexity of $\Theta(m.p)$ for writing to memory (m) exposed by all the processes (p). Figures 16 and 17 show the performance of shift and transpose communication primitives respectively. These benchmarks are expected to provide $\Theta(p)$ time complexity with a constant message size. The observed trends do not strictly follow this model. With scale, increased contention is observed on the links due to the over-subscription of links by intra-job and inter-job processes. The adaptive routing algorithm is limited, as it still uses the links in dimension order.

C. Performance Evaluation with Application Kernels

This section presents a performance evaluation of ARMCI-Gemini using application kernels LU decomposition, and

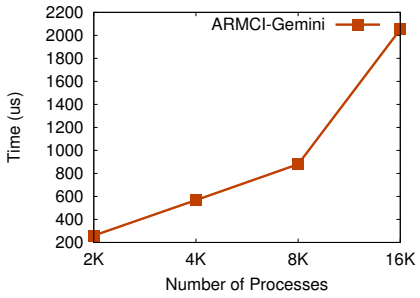

Fig. 16. Shift Performance, 512 Bytes

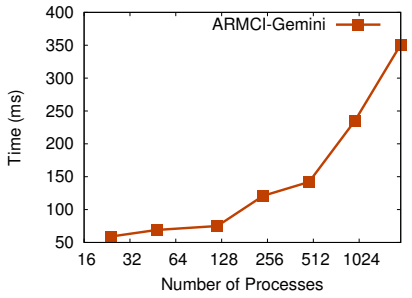

Fig. 17. Transpose Performance, 512 Bytes

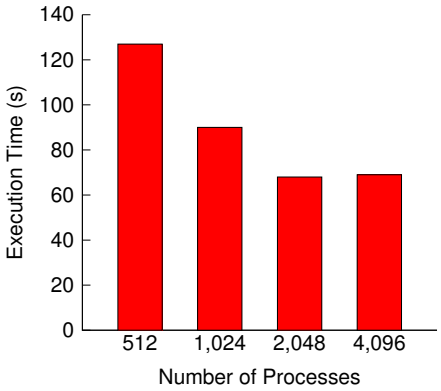

Fig. 18. LU Decomposition Performance, Dimension = 32768

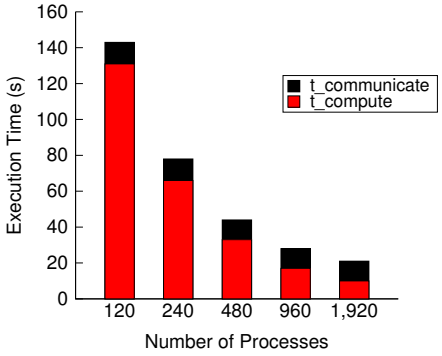

Fig. 19. Lennard Jones Simulation Performance

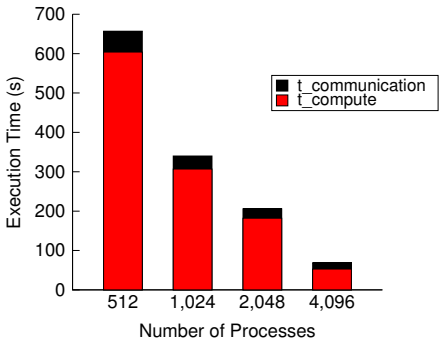

Fig. 20. Smooth Particle Hydrodynamic Performance

Lennard Jones Simulation.

1) LU Decomposition: The LU decomposition is an important kernel to factor a matrix A of dimension N in L and U matrices respectively. This factorization is useful in solving the set of linear equations. A dense LU decomposition kernel with 2D block data distribution. The broadcast step of the application kernel is performed using one-sided non-blocking put operations. These operations provide zero-copy transfer to processes in the row and column and communication/computation overlap, with the use of RDMA. In LU decomposition, communication volume is $\Theta\sqrt(p)$ with 2D decomposition and the overall computation reduces proportion to $\Theta(p)$. Figure 18 shows the performance of LU decomposition with strong scaling of 32768 matrix size and block distribution. The overall time decreases from 512 processes to 1024 processes (130s to 90s) but flattens out at 2048 processes. With strong scaling, we observe that the overall computation/process decreases and as a result the computation efficiency bottoms out.

2) Lennard Jones Simulation: The Lennard Jones potential is a molecular dynamics algorithm that approximates the interaction between a pair of neutral atoms or molecules. There are 3 classes of parallelization: Atom, Force and Spatial Decomposition. An implementation based on force decomposition and Global Arrays (GA) is used. The entire force matrix ($N \cdot N$) is divided into multiple blocks ($m \cdot m$) for dynamic load balancing, where m is the block size and N is the total number of atoms/particles. The force between two atoms/particles is approximated by Lennard Jones potential energy function. The primary communication primitives involved are load balance counters, put operations and synchronization.

The simulation uses 8192 atoms. At this problem size, the load balancing algorithm using counters (atomic fetch-and-add) provides linear scaling in computation(t_compute). The communication volume updated by each process is dependent on the overall problem size. The update (put operations) operations dominate the communication time(t_communicate) and remain constant with scale.

D. Performance Evaluation with Smooth Particle Hydrodynamics (SPH) Application

The SPH application is based on a Lagrangian approach which uses particles to simulate the behavior of fluid flow in complex geometries. The parallel implementation of the SPH algorithm is based on spatial decomposition of the rectangular simulation domain into subregions. The number of subregions is equal to the number of available processors and each processor is responsible for all the particles in the spatial region assigned to it. The communication template of SPH represents grid based exchange - computation time complexity is inversely proportional to Θ(p) and communication time is proportional to $\Theta\sqrt(p)$.

In the experimentation, 16 million particles are used for simulation. Figure 20 demonstrates the performance of SPH using up to 4096 processes. The t_compute and t_communication follows the model from 512 to 2048 processes. A super-linear speedup is observed from 2048 to 4096 processes. At 4096 processes, the grid fits in the cache resulting in lesser memory traffic. The t_communication still follows the expected model. The consensus in analytical model and empirical results demonstrates the efficacy of ARMCI-Gemini.

E. Discussion

An important observation is that the overhead of ARMCI library is negligible. Hence, the ARMCI-Gemini is used as an intermediate substrate between Global Arrays and DMAPP layer. An advantage of this approach is that the Global Arrays layer remains same between different implementations.

Another observation is that the performance of Global Arrays based applications is not compared against equivalent MPI based applications in this paper. This is primarily because MPI based codes typically use a static partitioning based algorithm with two sided implementations. GA based applications are more appropriate for algorithms, where the algorithms does not have a regular communication patterns. A comparison of these implementations typically leads to incorrect conclusions, because the programming models facilitate different algorithms.

V. Related Work

Designing scalable communication subsystems has been of interest to many research groups with primary focus on two-sided communication with MPI [1], [2].

Multiple studies have been undertaken on designing scalable communication subsystems on Cray Gemini Interconnect. Scalable message passing for Cray Gemini Interconnect has been proposed [19]. A similar study for Charm++ has been undertaken [20]. However, these studies have focused on message passing using uGNI, while ARMCI-Gemini is specifically designed and implemented for PGAS models. GASNet, the communication subsystem for Berkeley UPC has a beta-version of uGNI implementation at the point of final version of the paper. Hence, a comparison with GASNet is not undertaken in this paper.

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 [21], [22], [23], [24], [25]. Efforts for scalable MPI design in other Interconnects such as Quadrics, Myrinet, High Performance Switch have also been performed [26], [27], [28]. However, none of the efforts above address the one-sided communication aspects, which are addressed in this article.

A significant body of research has focused on performance evaluation at the user-access layer for Cray Gemini Interconnect [11], [13]. However, the efforts are for understanding the communication performance of user-access layers, and not designing a communication subsystem. The proposed framework in this paper uses these studies to design scalable communication protocols for ARMCI.

VI. Conclusions and Future Work

The major contributions of the paper are:

- A detailed discussion on Cray Gemini userspace libraries and applicability of these libraries for PGAS models. The paper presents a performance and productivity argument in using the userspace libraries.
- An exploration of solution space using Distributed Shared Memory Application (DMAPP) interface is undertaken and design is proposed for native one-sided communication runtime system using ARMCI (The proposed design is referred as ARMCI-Gemini). The communication protocols which leverage the network concurrency and communication/computation overlap protocol for accumulate operations are presented. The time-space complexities of communication protocols for contiguous, uniformly non-contiguous datatype communication, atomic memory operations and memory synchronization are analyzed.
- The proposed design (ARMCI-Gemini) is implemented and the performance is evaluated by using up to 8192 processes. The efficacy of ARMCI-Gemini is demonstrated using communication primitives (contiguous, non-contiguous, atomic memory operations), communication benchmarks (shift, transpose), application kernels (LU, Lennard Jones simulation) and an application (Smooth Particle Hydrodynamics). The proposed approach can achieve a get latency of 1.3 μs and a peak bandwidth of 6 GB/s, incurring negligible overhead on DMAPP userspace library.

The performance of ARMCI-Gemini on larger scale for understanding the fault tolerance and energy efficiency aspects of our design is an ongoing part of our research. We are working closely with Cray to leverage this study for their future generation of network architectures. We plan to release the proposed implementation for use by the open source community in future releases of ARMCI.

VII. Acknowledgment

We would like to acknowledge Dr. Ryan Olson at Cray, Inc for invaluable insights on Cray Gemini Interconnect and DMAPP library.

References

[1] 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.

[2] 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.

[3] W. Joubert and S.-Q. Su, "An Analysis of Computational Workloads for the ORNL Jaguar System," in *International Conference on Supercomputing*, 2012, pp. 247–256.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] 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, pp. 175–184.

[10] A. Vishnu, M. Krishnan, and D. K. Panda, "A Hardware-Software Approach to Network Fault Tolerance wwith InfiniBand Cluster," in *International Conference on Cluster Computing*, 2009, pp. 479–486.

[11] 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.

[12] "NERSC Hopper Phase II System," http://www.nersc.gov/users/computational-systems/hopper.

[13] 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.

[14] 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.

[15] R. W. Numrich and J. Reid, "Co-array Fortran for Parallel Programming," *SIGPLAN Fortran Forum*, vol. 17, pp. 1–31, August 1998.

[16] K. C. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, "Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters: Case studies with Scatter and Gather," in *IPDPS Workshops*, 2010, pp. 1–8.

[17] A. Vishnu, A. R. Mamidala, H.-W. Jin, and D. K. Panda, "Performance Modeling of Subnet Management on Fat Tree InfiniBand Networks using OpenSM." in *Proceedings of First International Workshop on System Management Techniques, Processes, and Services, held in conjunction with IPDPS'07*, 2005.

[18] M. Bruggencate and D. Roweth, "DMAPP: An API for One-Sided Programming Model on Baker Systems," *Cray Users Group (CUG)*, aug. 2010.

[19] 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.

[20] 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.

[21] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. P. Kini, D. K. Panda, and P. Wyckoff, "Microbenchmark performance comparison of high-speed cluster interconnects." *IEEE Micro*, vol. 24, no. 1, pp. 42–51, 2004.

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

[23] S. Sur, A. Vishnu, H.-W. Jin, W. Huang, and D. K. Panda, "Can memory-less network adapters benefit next-generation infiniband systems?." in *Hot Interconnects*, 2005, pp. 45–50.

[24] 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.

[25] B. Chandrasekaran, P. Wyckoff, and D. K. Panda, "Miba: A micro-benchmark suite for evaluating infiniband architecture implementations." in *Computer Performance Evaluation / TOOLS*, 2003, pp. 29–46.

[26] 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.

[27] 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.

[28] R. Sivaram, R. K. Govindaraju, P. H. Hochschild, R. Blackmore, and P. Chaudhary, "Breaking the Connection: RDMA Deconstructed," in *Hot Interconnects*, 2005, pp. 36–42.