Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction

Richard L. Graham, Devendar Bureddy
Pak Lui, Hal Rosenstock and Gilad Shainer
Mellanox Technologies, Inc.
Sunnyvale, California
Email: richardg, devendar, pak, hal, shainer
@mellanox.com

Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky,
Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen,
Alexander Shpiner, Oded Wertheim and Eitan Zahavi
Mellanox Technologies, Inc.
Yokneam, Isreal
Email: gil, gdror, miked, sashakot, vladimirk, lionl, alexm,
tamir, alexshp, oddedw, eitan @mellanox.com

***Abstract*—Increased system size and a greater reliance on utilizing system parallelism to achieve computational needs, requires innovative system architectures to meet the simulation challenges. As a step towards a new network class of co-processors - intelligent network devices, which manipulate data traversing the data-center network, this paper describes the SHArP technology designed to offload collective operation processing to the network. This is implemented in Mellanox's SwitchIB-2 ASIC, using in-network trees to reduce data from a group of sources, and to distribute the result. Multiple parallel jobs with several partially overlapping groups are supported each with several reduction operations in-flight. Large performance enhancements are obtained, with an improvement of a factor of 2.1 for an eight byte *MPI_Allreduce()* operation on 128 hosts, going from 6.01 to 2.83 microseconds. Pipelining is used for an improvement of a factor of 3.24 in the latency of a 4096 byte *MPI_Allreduce()* operations, declining from 46.93 to 14.48 microseconds.**

I. Introduction

In recent decades the quest for a steady increase in High Performance Computing (HPC) capabilities has caused significant changes in the architecture of such systems, to meet the ever growing simulation needs. Many architectural features have been invented to support this demand. This has included the introduction of vector compute capabilities to single processor systems, such as the CDC Star-100[1] and the Cray-1[2], followed by the introduction of small-scale parallel vector computing such as the Cray-XMP[3], custom-processor-based tightly-coupled MPPs such as the CM-5[4] and the Cray T3D[5], followed by systems of clustered commercial-off-the-shelf micro-processors, such as the Dell PowerEdge C8220 Stampede at TACC[6] and the Cray XK7 Titan computer at ORNL[7]. For a decade or so the latter systems relied mostly on Central Processing Unit (CPU) frequency up-ticks to provide the increase in computational power. But, as a consequence of the end of Dennard scaling[8], the single CPU frequency has plateaued, with contemporary HPC cluster performance increases depending on rising numbers of compute engines per silicon device to provide the desired computational capabilities. Today HPC systems use many-core host elements that utilize, for example, X86, Power, or ARM processors, General Purpose Graphical Processing Units (GPGPUs) and Field Programmable Gate Arrays (FPGAs)[9], to keep scaling the system performance.

Much of the focus on increasing system capabilities has been on increasing micro-processor and compute accelerator capabilities. This may be through increased computational abilities, e.g. adding vector processing facilities, raw hardware capabilities, e.g. increased clock frequency, of individual components, the increase in the number of such components, or some combination thereof. Network capabilities have also increased dramatically over the same period, with changes such as increases in bandwidth, decreases in latency, and communication technologies like InfiniBand RDMA that offload processing from the CPU to the network. However, the CPU has remained the focal point of system data management.

As the number of compute elements grows, and the need to expose and utilize higher levels of parallelism grows, it is essential to reconsider system architectures, and focus on developing architectures that lend themselves better to providing extreme-scale simulation capabilities. This includes support for processing data at the appropriate places in the system and reducing the amount of data that is moved between memory locations [10], [11]. Consequently, modern HPC architectures should investigate alternative specialized system elements that distribute the data manipulation, as appropriate, rather than having all data processing handled by a local or remote CPU.

Collaboration between all system devices and software to produce a well-balanced architecture across the various compute elements, networking, and data storage infrastructures is known as the Co-Design architecture. Co-Design exploits system efficiency and optimizes performance by ensuring that all components serve as co-processors in the system, creating synergies between the hardware and the software, and between the different hardware elements within the system. The capabilities described in this paper are directed towards such an architecture. The concept of Co-Design first presented

©2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

by DeMicheli [12] within the environment of chip design and later expanded to distributed systems and networks [13].

Mellanox focuses on CPU offload technologies performed by the network as data moves through it, be it on the Host Channel Adapter or the switch. This frees up CPU cycles for computation, reduces the amount of data transferred over the network, allows for efficient pipelining of network and computation, and provides for very low communication latencies. To accomplish a marked increase in application performance, there has been an effort to optimize often used communication patterns, such as collective operations, in addition to the continuous improvements to basic communication metrics, such as point-to-point bandwidth, latency, and message rate.

A key requirement for the success of new network features is the ability of applications to use the feature without requiring application-level modifications. In order to achieve this, it is a requirement to expose new features via Application Programmer Interfaces (API) which are ubiquitous in HPC, such as the Message Passing Interface (MPI)[14]. With the emergence of the OpenSHMEM[15] specification it is preferable to support this API too. In this paper we focus on the optimization of reduction operations. The MPI standard defines several types of collective operations that result in data reductions, including blocking and nonblocking variants of *MPI_Barrier()*, *MPI_Allreduce()*, *MPI_Reduce()*, *MPI_Reduce_scatter()*, *MPI_Scan()* and *MPI_Exscan()*. The OpenSHMEM specification currently defines blocking *shmem_barrier_all()*, *shmem_barrier()*, and the reduction operations *shmem_dt_op_to_all()*, where dt stand for a data type, and op for the reduction operations. The supported capabilities described in this paper are generic, however they are targeted at supporting these two very important APIs.

This paper describes the Scalable Hierarchical Aggregation Protocol (SHArP) introduced to greatly decrease the latency of reduction operations. SHArP defines a protocol for reduction operations, which is instantiated in Mellanox's SwitchIB-2 device, providing support for small data reduce and allreduce, and for barrier. It optimizes these often used global operations by performing the reduction operations on the data as it traverses a reduction tree in the network, reducing the volume of data as it goes up the tree, instead of using a CPU-based algorithm where the data traverses the network multiple times. The technology enables the the manipulation of data while it is being transferred within the data center network, instead of waiting for the data to reach the CPU to operate on this data. The wide reduction trees used provide a highly scalable algorithm, reducing the latency of an eight byte data reduction on a system of 128 hosts by a factor of 2.1. The effect of this optimization on overall application performance depends on the frequency of using such calls, as well as the the skew in the collective initiation across the group of participating processes. The greater the skew, the less pronounced the impact. However, the later is true of for any aggregation algorithm, whether implemented in hardware or software.

As described in section II, this is not the first time support for aggregation is provided within the network. However, what is unique to this design is the emphasis on scale, with support for large radix reduction trees and simultaneous overlapping running jobs, each with multiple outstanding aggregation operations.

In subsequent sections we will describe previous work, the SHArP abstraction, how it is implemented in SwitchIB-2, and provide some experimental data to demonstrate the effectiveness of this approach in increasing system performance, and making available more CPU cycles for computation.

II. Previous Work

In the past extensive work has been done on improving performance of blocking and nonblocking barrier and reduction algorithms.

Algorithmic work performed by Venkata et al. [16] developed short vector blocking and non blocking reduction and barrier operations using a recursive K-ing type host-based approach, and extended work by Thakur [17]. Vadhiar et al. [18] presented implementations of blocking reduction, gather and broadcast operations using sequential, chain, binary, binomial tree and Rabsnseifner algorithms. Hoefler et al. [19] studied several implementations of nonblocking *MPI_Allreduce()* operations, showing performance gains when using large communicators and large messages.

Some work aimed to optimize collective operations for specific topologies. Representative examples are ref. [20] and [21], which optimized collectives for mesh topologies, and for hypercubes, respectively.

Other work presented hardware support for performance improvement. Conventionally, most implementations use the CPU to setup and manage collective operations, with the network just used as a data conduit. However, Quadrics[22] implemented support for broadcast and barrier in network device hardware. Recently IBM's Blue Gene supercomputer included network-level hardware support for barrier and reduction operations. Its preliminary version Blue Gene/L [23] which uses torus interconnect [24], provided up to twice throughput performance gain of all-to-all collective operations [25], [26]. On a 512 node system the latency of the 16 byte *MPI_Allreduce()* the latency was 4.22 μ-seconds. Later, a message passing framework DCMF for the next-generation supercomputer Blue Gene/P was introduced [27]. MPI collectives optimization algorithms for this generation of Blue Gene were analyzed in [28]. The recent version Blue Gene/Q [29] provides additional performance improvements for MPI collectives [30]. On a 96,304 node system, the latency of a short allreduce is about 6.5 μ-seconds. IBM's PERCS system [31] fully offloads collective reduction operations to hardware. Finally, Mai et al. presented the NetAgg platform [32], which uses in-network middleboxes for partition/aggregation operations, to provide efficient network link utilization. Cray's Aries network [33] implemented 64 byte reduction support in the HCA, supporting reduction trees with a radix of up to 32. The eight byte *MPI_Allreduce()* latency for about 12,000 process with 16 processes per host was close to ten u-seconds.

Several API's have been proposed for offloading collective operation management to the HCA. This includes the Mellanox's CORE-Direct [34], protocol, Portal 4.0 triggered operations [35], and an extension to Portals 4.0 [36]. All these support protocols that use end-point management of the collective operations, whereas in the current approach the end-points are involved only in collective initiation and completion, with the switching infrastructure supporting the collective operation management.

III. Aggregation Protocol

A goal of the new network co-processor architecture is to optimize completion time of frequently used global communication patterns and to minimize their CPU utilization. The first set of patterns being targeted are global reductions of small amounts of data, and include barrier synchronization, and small data reductions.

SHArP provides an abstraction describing data reduction. The protocol defines aggregation nodes (ANs) in an aggregation tree, which are basic components of in-network reduction operation offloading. In this abstraction, data enters the aggregation tree from its leaves, and makes it's way up the tree, with data reductions occurring at each AN, with the global aggregate ending up at the root of the tree. This result is distributed in a method that may be independent of the aggregation pattern.

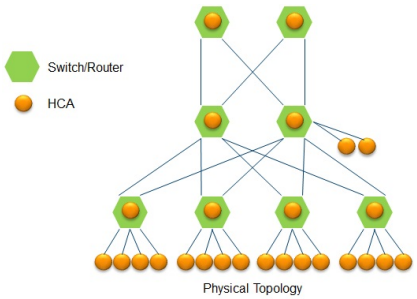
Much of the communication processing of these operations is moved to the network, providing host-independent progress, and minimizing application exposure to the negative effects of system noise. The implementation manipulates data as it traverses the network, minimizing data motion. The design benefits from the high degree of network-level parallelism, with the high-radix InfiniBand switches to use shallow reduction trees.

The aggregation protocol is described in the following subsections. Data enters the aggregation tree at its leaves, with the aggregation nodes operating on the data as it travels up the tree to the root. Aggregation groups are used to minimize the aggregation data path.

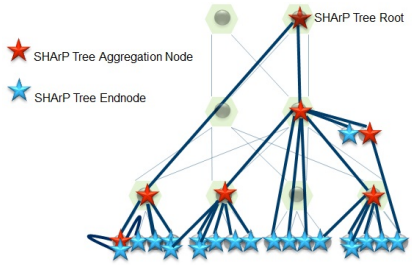
A. Aggregation Nodes

The aggregation node is a logical construct, and specifically a node in an aggregation tree. It accepts data from its children, reduces the data, and if appropriate, forwards the result to its parent. If the node is defined as the root, it starts distributing the result, instead of forwarding it up the tree. The operations supported by the protocol are those that keep the volume of the resulting data the same as that coming in from an individual child in the tree. This supports barrier-synchronization, with zero size user data, as well as vector reductions, with operations such as a summation.

An aggregation can be implemented in different ways. It can be instantiated, as a process running in a server connected to the cluster, as a process running in a switch device, or as part of the switch hardware.



(a) Physical Network Topology



(b) Logical SHArP Tree. Note that in the SHArP abstraction an Aggregation Node may be hosted by an end-node.

Fig. 1: SHArP Tree example.

B. Aggregation Trees

The aggregation tree defines the reduction pattern for data entering from the end-nodes, with the result ending up in the root of the tree. ANs are connected in a logical tree topology.

Figure 1 shows an example of a SHArP tree allocation over the physical network topology. Figure 1a shows an example of physical network fat tree topology, and Figure 1b is the SHArP tree allocation over this topology. Generally, since the SHArP tree is logical, it can be created over any topology. The optimization of tree allocation over given topology is out of scope of this paper.

SHArP end-nodes (denoted in the figure by blue stars) are connected to ANs (denoted in the figure by red stars), and, together with the ANs, define a SHArP tree. As seen in the figure, ANs are usually implemented in a switch, but also can be implemented in a host. In addition, the connections between ANs are logical and, hence, do not have to follow the physical topology. Moreover, the end-nodes are not necessarily connected to the physically nearest switch, or AN, in a SHArP tree. One node of the tree is defined as a root, and that defines the parent-children hierarchy for the ANs in the tree.

Generally, each AN can participate in several trees, however each reduction operation can use only a single tree at a time. Above the SHArP level,l single reduction can be split into multiple smaller reduction operations, each performed on the same or different trees.

The protocol does not define the data transport, so that communication can occur using a range of transports, such as RDMA-enabled protocols like InfiniBand or RDMA over Converged Ethernet (RoCE). It also does not handle packet

loss or packet reordering, requiring a reliable transport which provides reliable, in-order delivery of packets to the upper layer.

Multiple trees are supported to better distribute the load over the system, and utilize available aggregation resources. No assumptions are made about the physical topology of the underlying network, and trees may overlay any physical network topology, such as a fat-tree, DragonFly+, and hypercube.

C. Aggregation Group

As a mechanism for improving system resource utilization, the implementation defines the concept of a group, which includes a subset of the hosts, that are attached to the leaf nodes of the tree. A group may be defined to include the subset of hosts spanning a communicator. When the group is formed, the AN creates a description of its children and its parent (if it is not the root) thus defining a sub-tree over which reductions will take. A given tree supports multiple groups, from a single job as well as from multiple jobs belonging to different users.

As a performance optimization, once a group has been defined and the corresponding sub-tree setup, this sub-tree may be trimmed, terminating it at the level is the sub-tree's where its width become one the rest of the way to the root of the tree. This is done to eliminate the need for all data to reach the root of the tree; rather, it only must reach the highest level in the tree needed to reduce that data. The detailed algorithm of distributed group creation is out the scope of the paper.

For example a SHArP group is created when a new MPI communicator is created. This group contains a subset of MPI processes that are members of this communicator, one MPI process per host, or perhaps one MPI process per socket. Each group member is responsible for on-host or on-socket data aggregation, before passing this aggregated data to the SHArP protocol.

To avoid potential deadlock in the collective operation, one must ensure end-to-end availability of SHArP resources, such as buffers, in the ANs. If this is not ensured, the simultaneous and uncoordinated execution of multiple reduction operations competing for the same AN resources may result in several reduction operations waiting for the availability of the same AN resources, allowing none to complete. Resource allocation methods are not covered in this paper. When the result arrives at its targets, the implementation ensures that none of the resources used in the reduction are still in use. This may be used for flow control purposes.

D. Aggregation Operations

An aggregation operation is performed with participation of each member of the aggregation group. To initiate such an operation, each member of the aggregation group sends an aggregation request message to its leaf aggregation node. The SHArP request message includes the aggregation request header followed by the reduction payload. Figures 2 and 3 show the format of the SHArP message.

The aggregation request header contains all needed information to perform the aggregation. This includes the data

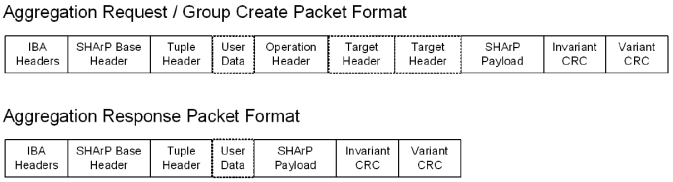


Fig. 2: SHArP Header schematic.

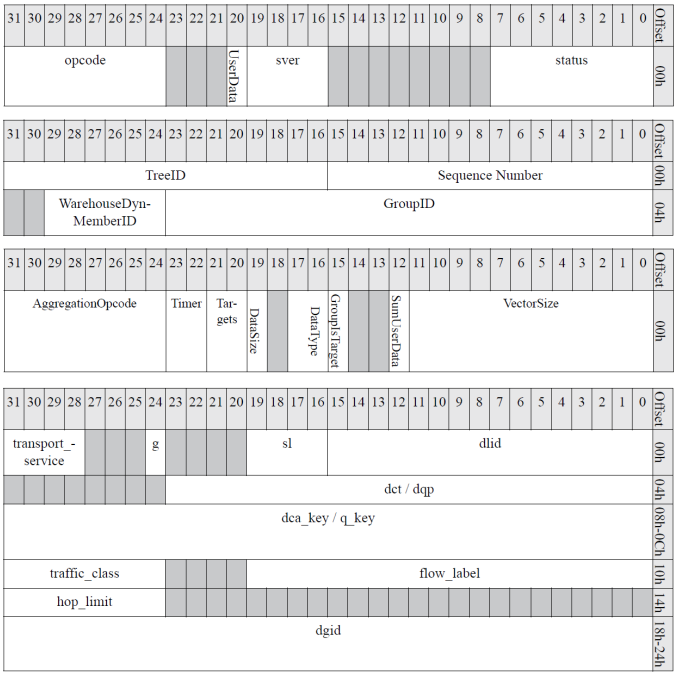


Fig. 3: SHArP Header including data type, size, and number of elements, as well as operation code.

description, i.e., the data type, data size, and number of such elements, and the aggregation operations to be performed, such as a min or sum operation.

An aggregation node receiving aggregation requests collects these from all its children and performs the aggregation operation once all the expected requests arrive. An internal data structure per collective operation at each AN is used to track the progress of a collective operation. These are allocated on the arrival of the first message associated with the collective operation, and freed once the operation has completed locally. The result of the aggregation operation is sent by the AN as a part of a new aggregation request to its parent aggregation node. The root aggregation node performs the final aggregation producing the result of the aggregation operation.

Upon completion of the aggregation operation, the root aggregation node forwards the aggregation result to the specified destinations. The destination may be one of several targets, including one of the requesting processes, such as in the case of *MPI_Reduce()*, all the group processes, such as in the case of an *MPI_Allreduce()* operation, or a separate process that may not be a member of the reduction group, such as in the case of big data MapReduce type of operations. An

aggregation tree can be used to distribute the data in the these cases. The target may also be a user-defined InfiniBand multicast address. It is important to note that while multicast data distribution is supported by the underlying transport, it provides an unreliable delivery mechanism. Any reliability protocol needed must be provided on top of this mechanism.

E. Fault Handling

Several types of errors may occur and include transport-level errors, end-node errors, and SHArP protocol errors. The main goal of the fault handling mechanism is to notify the host-processes of such errors, under the assumption that some data is lost, and therefore the aggregation operation can't complete successfully. When this happens, the allocated AN resources are freed, as there is no guarantee that the specific error can be dealt with, and the end-user processes are notified, so the application can make a decision how to proceed, whether it can recover from the error, or whether it must abort.

With all types of errors, a management entity, known as the Aggregation Manager (AM), is notified of the error, if possible. The AM will attempt to notify the end-node processes of the error, and then revoke the SHArP resource allocation associated with the running job. Both in-band SHArP monitoring capabilities, as well as InfiniBand network monitoring capabilities are used to detect errors. Since SHArP is also intended to support MPI and SHMEM communication libraries, which do not bound the duration of an aggregation operation, timeouts can't be relied on for error detection. Description of the software infrastructure supporting the hardware capabilities is beyond the scope of this paper.

SHArP errors, such as tree connection failure or errors within the Target Channel Adapter (TCA), are detected by the SHArP hardware implementation. The AM is notified as the result of such error detection.

Transport specific errors, such as data transmission errors, link errors, and switch failure, are handled by the transport layer and its monitoring infrastructure. If the transport layer can recover from such errors, and data transmission proceeds, the AM is not notified of such errors.

Errors in the end-nodes, such as connection failure and process failure, are detected by the leaf switches, when they attempt to communicate with the host, and result in AM notification.

F. SwitchIB-2-Based Aggregation Support

The implementation of support for the switch-based reduction operations is based on the SHArP protocol described in Section III. The aggregation node logic is implemented as an InfiniBand TCA integrated into the switch ASIC. The transport used for communication between ANs and between AN and hosts in the aggregation tree is the InfiniBand Reliable Connection (RC) transport. The results are distributed from the root to the leaf nodes, or hosts, down the tree, or to a target InfiniBand Multi-cast group. Details are provided below.

Mellanox's SwitchIB-2 implements support for SHArP over the InfiniBand network. The SHArP aggregation node is integrated into the switch ASIC, which adds to its ability to provide the best performance in terms of bandwidth and latency. As illustrated in Figure 4, an internal logical port connects an InfiniBand SHArP TCA to the switch. This TCA provides an InifinBand transport termination point, which serves as the mechanism for targeting data to be manipulated by the aggregation logic, and as the source for the result of such aggregation, as required by the aggregation protocol. The TCA supports both Reliable Connection (RC) transport to enable reliable delivery of data through the aggregation tree as well as Unreliable Datagram (UD) transport to enable Multicast distribution of the aggregation result. Data is sent in the direction of the root, when aggregation is taking place, using the RC transport. The results may be distributed from the root down the tree using the RC transport, and to a UD multicast group using the UD transport. A high performance hardware implementation of the aggregation node functionality is embedded into the TCA hardware.

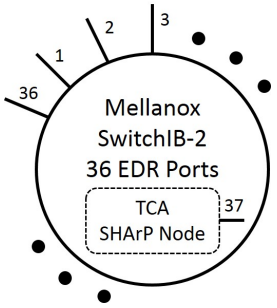


Fig. 4: SHArP Node implemented as a virtual TCA within the SwitchIB-2.

The aggregation node implementation includes a high performance Arithmetic Logic Unit (ALU), used to perform the aggregation operations supported by the aggregation node. It can operate on 32- and 64-bit signed and unsigned integers and floating point data. The supported operations include sum, min and max, MPIs MinLoc and MaxLoc, bitwise OR, AND, and XOR, which include all the operations, with the exception of the product, needed to support the MPI standard and the OpenSHMEM specification.

While floating point summation and multiplication operations are both commutative, they are not associative. That is, (a+b)+c not necessarily equal to a+(b+c). While performing a non-commutative reduction operation on floating point operands, the implementation needs to take care of the order of execution to enable repeatable operation results. As an example, $10^{20} - (10^{20} + \epsilon) = 0$ is not the same as $(10^{20} - 10^{20}) - \epsilon = -\epsilon$. Therefore, the operation can not be performed in the order the aggregation requests arrive at the aggregation node, since this order is non-deterministic. Requests are collected in the TCA, with the reduction performed only after all operands are available, in a predetermined fixed order. SwitchIB-2 implements a predictable operation ordering to enable repeatable results regardless of the order of arrival of the aggregation requests.

An important design goal for the SwitchIB-2 SHArP implementation is that this functionality be useful in large-scale production environments, with either few large running jobs, or many smaller jobs. While such jobs may not share compute nodes, they could still use some of the same switches, and it is essential to minimize the impact of a running job on the other ones. To reduce the impact of a collective operation on other data flows, and avoid switch buffer resources being used for long periods of time, which waiting for data from a late arriving process to reach the reduction node, the aggregation support is implemented in a separate unit, the TCA, within the ASIC and does not require dedicated virtual lanes. The implementation support for up to 64 trees per aggregation node, a large number of groups and hundreds of outstanding aggregation operations enables many applications running at the same time to run at the same time using overlapping switch resources. Multiple reduction groups per job are also supported each being able to perform several simultaneous reduction operations efficiently.

Access to the SHArP trees from the host is controlled by the privileged SHArPd which obtains the list of allocated resources from the Aggregation Manager (AM). The SHArPd, in lieu of RDMA CM running on the switch, is involved in establishing the RC connections between the hosts and the leaf switches, and enforces the resource allocation provided by the AM. In addition, InfiniBand Partition Keys (pkey's) are used for traffic isolation. For each job, the AM determines the switch resources allocated to the job, and passes this information to all relevant AN's, which track this resource, and the amount of resources in current use by the application, not allowing it to exceed the allocation.

IV. MPI Implementation

In the MPI implementation a SHArP group is mapped to an MPI communicator. New group formation includes new group formation, group trim operation, and a query for the resource allocation for the group. The cost of these operations are logarithmic in the tree radix. Group formation and trim are accelerated by the Aggregation Nodes.

The implementation uses a non-blocking interface to post a work request to the RC QP connected to a laef switch, to locally initiate a collective operation. A *MPI_Barrier()* is initiated with a SHArP barrier, and *MPI_Reduce()* and *MPI_Allreduce()* are initiated with an allreduce SHArP work request. Blocking and non-blocking collective operations are implemented using the same interfaces. The *MPI_Reduce()* is implemented in this manner, for ease of tracking credits, with the result discarded by all but the root of the communicator. Collective operation completion is detected by a completion event received after the result is delivered to the QP. The current implementation uses the aggregation tree to distribute the results to all members of the communicator.

The MPI implementation keeps track of the number of outstanding collective operations posted to the network, as well as the amount of buffer space in use. New collective operations are initiated, only if resources are available, otherwise operation initialization is delayed.

V. Experiments

To study of the switch-based SHArP collective acceleration hardware, the performance of the collective operations was measured as a function of message size and number of hosts, and compared with data obtained using optimized host-based methods.

A. System Configuration

The test system included 128 node dual socket hosts, with each processor being a 2.6 GHz Intel E5-2697 v3 Haswell processor. Mellanox's SwitchIB-2 switches, with support for the SHArP protocol, and ConnectX-4 Host Channel Adapters were used.

Mellanox OFED version 3.2-1.0.1.1 of the InfiniBand verbs was is use, with Red-Hat Linux version 7.2 operating system. A pre-release version of Mellanox's subnet manager with SHArP support was used. Pre-release versions of the Aggregation Manager, which manages the aggregation control-path, and SHArP Daemon, which handles host-local control-path management were also used. A pre-release version of SwitchIB-2 firmware was used, with SHArP control-path support.

Since the latencies of the barrier and small-sized data reduction operations are becoming rather small, effects of system noise have a measurable effect on the performance of these collective operations. As such, several default system settings were modified to reduce the effect of the system activity, and the Central Processing Unit (CPU) entering one of the C-states. The kernel/OS settings include the command lline options intel_pstate=disable, isolcpus=1, nohz_full=1 and rcu_nocbs=1. They also include sysctl vm_-stat_interval=1200, sysctl kernel.numa_balancing=0, sysctl kernel.nmi_watchdog=0 and disable THP: echo never ¿ /sys/kernel/mm/transparent_hugepage/enabled. The BIOS setting changes include disabling the SMI interrupts including ''Processor Power and Utilization Monitoring" and ''Memory Pre-Failure Notification". In addition disable C-states, set static max performance, disable turbo-boost and disable hyper-threading.

B. Communication Libraries

A pre-release version of HPC-X[37] is used. The MPI collective FCA[38] library version 3.4 is used, with initial SHArP support. Some comparison runs were performed using MVAPICH-2[39] version 2.2a.

C. Performance Tests

A low-level verbs test, and the OSU collective latency test version 5.2 [40] for MPI_Barrier() and MPI_Allreduce() were used. All tests were run with a single process per host.

The basic verbs-level tests consisted of a tight loop around individual collective operations, such as a SHArP-level barrier. This consists of two steps: 1) posting the SHArP barrier

message to the HCA, and 2) polling the SHArP completion queue for completion of the reduction operation. In a pipelined algorithm, a single MPI-level message was fragmented into multiple SHArP messages which were simultaneously in flight.

In addition, the impact of using the SHArP based aggregation support on the OpenFOAM [41] application is studied.

VI. MICRO-BENCHMARK RESULTS

A. Native SHArP Measurements

Table I present the results of SHArP-based latency measurements for barrier and allreduce operations. In all of these measurements the hosts are equally distributed across eight leaf switches configured in a two-level fat tree topology with 100Gbps links.

TABLE I: Native SHArP Allreduce() average latency (μ-seconds). Optimized system with turbo-mode off.

	Number of Hosts		
Message Size [B]	32	64	128
0	2.59	2.57	2.63
8	2.66	2.68	2.79
16	2.72	2.70	2.86
32	2.72	2.85	2.92
64	2.89	2.92	3.04
128	3.06	3.10	3.25
256	3.89	3.99	4.21

Table II compares the performance of the SHArP-based barrier and allreduce collectives on a system tuned for reduced system activity (Opt), as described in section V, and a system configured with default settings (Unopt). When 64 hosts are used, the impact on performance is relatively small, on the order of 5%. However, at 128 nodes the impact isin the range of 10 to 20%.

TABLE II: Native Allreduce() and Barrier() average latency (μ-seconds). One process per host. Comparison of optimized system to out-of-the-box system settings.

Message Size [B]	64 Hosts			128 Hosts		
	Opt	Unopt	% Diff	Opt	Unopt	% Diff
0 (Barrier)	2.57	2.70	5.0	2.63	3.17	20.5
8	2.68	2.85	6.3	2.79	3.43	22.9
16	2.70	2.79	3.3	2.86	3.15	10.1
32	2.85	2.90	1.7	2.92	3.36	15.1
64	2.92	3.06	4.8	3.04	3.45	13.5
128	3.10	3.24	4.5	3.25	3.68	13.2
256	3.99	4.02	0.8	4.21	4.82	14.5

B. MPI-Level SHArP Measurements

Table III reports the latencies of *MPI_Allreduce()* and *MPI_Barrier()* compared with native SHArP data. The MPI integration increases latency by two to four tenths of a μ-second.

Table IV compares the performance of *MPI_Barrier()* and *MPI_Allreduce()* from several different implementations at

TABLE III: A comparison of *MPI_Allreduce()* and Native Allreduce() average latencies (μ-seconds) on optimized system. The latency difference (Diff) is in units of μ-seconds.

	Number of Hosts					
Size [B]	32		64		128	
	MPI	Native	MPI	Native	MPI	Native
0	2.83	2.59 (0.24)	2.83	2.57 (0.26)	2.88	2.63 (0.25)
8	2.95	2.66 (0.29)	3.10	2.68 (0.42)	3.15	2.79 (0.36)
16	2.89	2.72 (0.17)	3.04	2.70 (0.34)	3.03	2.86 (0.17)
32	3.01	2.72 (0.29)	3.17	2.85 (0.32)	3.24	2.92 (0.32)
64	3.06	2.89 (0.17)	3.24	2.92 (0.32)	3.24	3.04 (0.20)
128	3.24	3.06 (0.18)	3.44	3.10 (0.34)	3.57	3.25 (0.32)

128 host count. The SHArP-based approach consistently outperforms a host-based algorithm, which uses InfiniBand multicast capabilities. At this scale the latency of the host-based methods are at least 50% than the SHArP-based methods. The difference from the MVAPICH-2 implementation is even larger, showing that the host-based HPC-X algorithms are efficient, and provide a fair comparison for the SHArP based algorithms.

TABLE IV: *MPI_Allreduce()* and *MPI_Barrier()* average latency (μ-seconds) for several implementations on 128 hosts, 1 process per host. The latency difference (Diff) is in % relative to SHArP based measurements.

Message Size [B]	HPC-X SHArP Based	HPC-X Host Based	MVAPICH-2 Host Based
0(barrier)	2.91	5.25 (80.4%)	11.47 (290%)
8	2.83	6.01 (112%)	11.90 (320%)
16	2.79	5.95 (56.9%)	11.27 (304%)
32	3.94	6.19 (57.1%)	11.69 (197%)
64	3.02	6.80 (125%)	12.03 (298%)
128	4.30	7.69 (78.8%)	14.08 (227%)

Table V shows *MPI_Allreduce()* latencies for message sizes that exceed the SHArP hardware maximum message size. This is achieved by posting multiple SHArP reduction operations, pipelining the reduction of message fragments. This improved performance, with a peak speedup of almost a factor of four at 2048 bytes.

TABLE V: *MPI_Allreduce()* average latency (μ-seconds) on 128 hosts, one process per host. Comparison of pipelined SHArP-based algorithm with host-based algorithm.

Message Size [B]	SHArP based	Host Based	SHArP improvement factor
8	2.76	5.82	2.11
16	2.76	5.91	2.14
32	2.86	6.04	2.11
64	3.01	6.76	2.25
128	3.24	7.37	2.27
256	3.50	8.99	2.57
512	4.06	11.11	2.74
1024	5.49	18.04	3.29
2048	8.44	33.61	3.98
4096	14.48	46.93	3.24

TABLE VI: *MPI_Allreduce()* average latency (μ-seconds) on 128 hosts, one process per host. Comparison of result deitributiong using RC down the reduction tree, UD multicast, and both with a host-based algorithm. Size is given in bytes.

size	RC Only	RC & UD (default)	UD Only	Host based
4	2.7	2.5	2.46	5.68
8	2.7	2.51	2.45	5.78
16	2.71	2.56	2.52	5.89
32	2.82	2.56	2.51	6.00
64	2.96	2.67	2.61	6.78
128	3.14	2.96	2.92	7.3
256	3.44	3.75	3.71	8.59
512	4.1	4.29	4.19	10.69
1024	5.51	5.5	5.43	18.67
2048	8.45	7.94	7.47	33.7
4096	14.62	13.31	11.54	46.89

VII. OpenFOAM Performance

Understanding the impact of the SHArP technology on application performance is important. To have an impact such applications must use either barrier orsmall reduction operations. Also, application load imbalance must be smaller than the duration of the collective operation to exhibit significant impact on application performance.

We have chosen to use the OpenFOAM [41] application for these measurements. The benchmark run was the Lid Driven Cavity Flow, using the icoFoam solver, with 1 million cells.

The tests were run using HPC-X and base Open MPI. HPC-X's collective *MPI_Allreduce()* algorithms are shared-memory aware. The collective operation used most frequently by OpenFOAM is an eight byte *MPI_Allreduce()* used by the conjugate gradient solver.

The host-based *MPI_Allreduce()* used by HPC-X is a highly optimized derivative of the recursive k'ing algorithm that leverages the InfiniBand hardware multicast capabilities. It also support a SHArP based version of *MPI_Allreduce()*. The default Open MPI implementation uses a more traditional recursive-doubling *MPI_Allreduce()*.

Table VII present the total run time for this test case, as a funciton of MPI implementation and of the number of processes. As the results show, using the SHArP capabilities improves overall application performance. When comparing to the implementation using the highly optimized host-based algorithm, overall application performance improves anywhere from half a percent to fifteen percent. Discarding the last two data points, since overall run time is increased, five percent improvement is the upper limit on performance improvement for this particular test case.

Comparing the performance of the test case using the SHArP capabilities to the more traditional *MPI_Allreduce()* implementation, we see a far larger performance improvement. Overall application performance is improved in the range of four to 50%, or four to 30%, when considering only the runs for which an increased number of processes being used reduces overall application run-time.

TABLE VII: OpenFOAM performance as a function of the MPI library implementation. The numbers in brackets indicate the % improvement in run-time when using SHArP capabilities.

Number of Processes	HPC-X with HCOLL no SHArP	Default Open MPI	HPC-X with HCOLL with SHArP
28	1892	1892(0%)	1892 (0%)
56	832	866 (4.09%)	824 (1.05%)
112	426	469 (9.96%)	424 (0.59%)
224	249	285 (14.3%)	248 (0.39%)
448	159	194 (22.4%)	155 (2.50%)
896	127	167 (31.4%)	121 (4.75%)
1792	118	180 (52.6%)	116 (1.65%)
2688	136	206(51.3%)	127 (7.18%)
3584	174		151 (15.2%)

VIII. Discussion

Small and medium data reduction operations are frequently used by many scientific applications. The need for rapid synchronization of many processes is also great, especially when using programming models that rely on explicit synchronization to maintain application consistency, such as OpenSHMEM. This is contrary MPI send-receive-like semantics where the synchronization is implicit, as part of the protocol. The SHArP capabilities are designed to provide very low network-level latency reduction and synchronization capabilities, thus addressing these needs. In addition, this capability is designed to minimize the software path for such operations.

To achieve scalable low-latency performance, the implementation takes advantage of the large switch radix, 36 ports in the case of SwitchIB-2, by mapping the SHArP reduction tree onto the underlying physical topology in such a way that minimizes the network costs between aggregation nodes. In particular, there is a preference to map connected aggregation nodes to directly connected switches, where possible, to minimize switch latency costs, as well as link utilization. This results in shallow, high-radix reduction trees, and while the experiments run for this paper were for a system with a topology of a two-level fat-tree, this approach is expected to be efficient for other topologies. The high-radix trees provide a high degree of reduction parallelism, while rapidly reducing the volume of data on the network. Thus, data traverses a small number of switches, often one switch per tree level. In the case of a fat-tree topology based on a 36-port SwitchIB-2 supporting a full bisection-bandwidth, the data volume is reduced by up to a factor of 18 at each level in the tree.

The particular configuration chosen for this paper's experiments uses eight leaf switches, with the 32, 64, and 128 host configurations differing only in the number of children each AN has at the first-level in the tree. As barrier data in table I shows, the native performance is almost constant as a function of the number of children when the only changes in the tree are the number of children the AN has. There is a weak dependence on the size of the reduction of 0.10 and 0.22 micro-seconds when the number of children at the first-level AN is increased from four to eight, and then from eight

to sixteen, respectively. This is due to the larger number of individual data reductions that are performed, as well as the added volume of data the ASIC must handle.

MPI-level comparisons are also provided, as this is the communication API used by the vast majority of parallel HPC applications. Table III compares the latencies for the MPI level reduction capabilities to the native capabilities. This table shows that the current MPI-level support generally adds overhead in the range of 200-400 nano-seconds. This overhead is expected to be independent of the size of the reduction group, with a dependence on message size when an inline send is used to initiate the reduction operation.

This paper relies to a certain degree on performance comparisons to host-based MPI-level reduction operations to highlight the performance advantages of the SHArP-based approach. It is important to demonstrate that the measuring stick indeed does provide a good comparison for such purposes. Table IV compares the MPI-level performance of *MPI_Barrier()* and *MPI_Allreduce()* of SHArP-based implementations, to an industry leading and highly optimized host-based methods supported by HPC-X, and to MVAPICH-2, a widely used InfiniBand-supporting MPI implementation. Both are considered high-performance implementations for InfiniBand-based systems. As table IV shows, HPC-X's SHArP-based implementation is at least fifty percent better than it's host-based approach. At 128 nodes, the implementation is at a factor of three to four times better than the ones supported by the MVAPICH-2 implementation. The SHArP-based approach does indeed provide exceptional performance, with the difference in latencies expected to grow as the system size grows.

Comparing SHArP estimated eight byte *MPI_Allreduce()* performance to that reported for the highly integrated BG/L and BG/P machines, SHArP latencies are a bit smaller than the BG/L latencies. The SHArP latency is comparable to that of the BG/P platform. However, the SHArP implementation supports an environment where jobs may share host and network resources, and does not require taking up virtual lanes to provide switch buffer resources for SHArP reductions. The BG/P per router latencies are quite a bit lower than those of the SwitchIB-2, but the latter supports a much wider tree node radix, with the latency of these small data reductions being weakly dependent on the number of input operands. This supports shallow reduction trees, thus providing low overall reduction latencies.

Even though the switch can perform reductions on payloads of up to 256 bytes, independent of the size of the individual elements, the ability of the device to handle multiple outstanding collective operations is used to pipeline reductions. This capability is used to support the reduction of larger data blocks, using software level message fragmentation and reassembly. As table V shows, this greatly increases the range of SHArP's utility. In this particular set of measurements, the benefit over the host-based approach continues to grow in relative proportions up to a size of 2048 bytes, where the latency of the SHArP-based approach is about a factor of four above an equivalent optimized host-based algorithm. Large improvements are shown over the full range of measurements, up to 4096 bytes.

Finally, as has been shown by the OpenFOAM test case, application run time is reduced by using the SHArP capabilities, with its impact being more pronounced with increased use of SHArP optimized collective operations. The more load balanced the applications are, the greater the impact on overall application performance. This is generally true for collective operations that aggregate data, with the SHArP based algorithms being such operations.

IX. Conclusions

This paper presents the SHArP architecture and its implementation in the SwitchIB-2 ASIC. The capability shows very good performance characteristics for small-sized data reductions, with industry leading performance. By mapping the aggregation trees well onto the underlying physical trees and taking advantage of the high degree of parallelism available at the network level, this approach is expected to scale well to very large systems. In addition, one can take advantage of the large amount of outstanding operations supported by the switches, by pipelining SHArP reduction operations to extend the range of reduction sizes that can benefit from these capabilities.

The current approach has more potential for improved performance over end-point (HCA) based approaches, because of the greater degree of aggregation concurrency available from the high-radix Mellanox InfiniBand switches. In addition, the data paths tend to be shorter, with the data not required to travel to the network edge to be reduced. For topologies where all switches are edge switches, such as hypercubes, the virtual nature of the AN implies that switch traversal does not imply paying the additional cost of a reduction at each switch traversal, thus enabling the use of shallow aggregation trees even for such network topologies.

References

[1] P. B. Schneck, *Supercomputer Architecture*. Boston, MA: Springer US, 1987, ch. The CDC STAR-100, pp. 99–117. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-7957-1_5

[2] R. M. Russell, "The cray-1 computer system," *Commun. ACM*, vol. 21, no. 1, pp. 63–72, Jan. 1978. [Online]. Available: http://doi.acm.org/10.1145/359327.359336

[3] M. C. August, G. M. Brost, C. C. Hsiung, and A. J. Schiffleger, "Cray x-mp: The birth of a supercomputer," *Computer*, vol. 22, no. 1, pp. 45–52, Jan. 1989. [Online]. Available: http://dx.doi.org/10.1109/2.19822

[4] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, "The network architecture of the connection machine cm-5 (extended abstract)," in *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '92. New York, NY, USA: ACM, 1992, pp. 272–285. [Online]. Available: http://doi.acm.org/10.1145/140901.141883

[5] R. Kessler and J. Schwarzmeier, "Cray t3d: a new dimension for cray research," in *Compcon Spring '93, Digest of Papers.*, feb 1993, pp. 176 –182.

[6] https://www.tacc.utexas.edu/systems/stampede.

[7] https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/.

[8] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 365–376.

[9] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with fpga-based computing," *Computer*, vol. 40, no. 3, p. 50, 2007.

[10] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Quantifying the energy cost of data movement in scientific applications," in *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2013, pp. 56–65.

[11] D. Tiwari, S. S. Vazhkudai, Y. Kim, X. Ma, S. Boboila, and P. J. Desnoyers, "Reducing data movement costs using energy-efficient, active computation on ssd," in *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, 2012.

[12] G. DeMicheli and M. Sami, *Hardware/software Co-design*. Springer Science & Business Media, 2013, vol. 310.

[13] F. Balarin, *Hardware-software co-design of embedded systems: the POLIS approach*. Springer Science & Business Media, 1997.

[14] http://www.mpi-forum.org.

[15] http://www.openshmem.org.

[16] M. G. Venkata, P. Shamis, R. Sampath, R. L. Graham, and J. S. Ladd, "Optimizing blocking and nonblocking reduction operations for multicore systems: Hierarchical design and implementation," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–8.

[17] R. Thakur and R. Rabenseifner, "Optimization of collective communication operations in mpich," *International Journal of High Performance Computing Applications*, vol. 19, pp. 49–66, 2005.

[18] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra, "Automatically tuned collective communications," in *In Proceedings of SC99: High Performance Networking and Computing*. IEEE Computer Society, 2000, p. 3.

[19] T. Hoefler, J. M. Squyres, and W. Rehm, "A case for nonblocking collective operations," in *In Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops*. Springer, 2006, pp. 155–164.

[20] M. Barnett, R. J. Littlefield, D. G. Payne, and R. A. van de Geijn, "Global combine on mesh architectures with wormhole routing," in *The Seventh International Parallel Processing Symposium, Proceedings, Newport Beach, California, USA, April 13-16, 1993.*, 1993, pp. 156–162. [Online]. Available: http://dx.doi.org/10.1109/IPPS.1993.262873

[21] V. W. Liu, C. Chen, and R. B. Chen, "Optimal all-to-all personalized exchange in d-nary banyan multistage interconnection networks," *Journal of Combinatorial Optimization*, vol. 14, no. 2, pp. 131–142, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10878-007-9065-5

[22] F. Petrini, S. Coll, E. Frachtemberg, and A. Hoisie, *Hardware-and-software-based collective communication on the Quadrics network.*, Jan 2001. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/975699

[23] A. Gara, M. A. Blumrich, D. Chen, G.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay *et al.*, "Overview of the blue gene/l system architecture," *IBM Journal of Research and Development*, vol. 49, no. 2, pp. 195–212, 2005.

[24] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus *et al.*, "Blue gene/l torus interconnection network," *IBM Journal of Research and Development*, vol. 49, no. 2/3, p. 265, 2005.

[25] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of all-to-all communication on the blue gene/l supercomputer," in *Proceedings of the 2008 37th International Conference on Parallel Processing*, ser. ICPP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 320–329. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2008.83

[26] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of mpi collective communication on bluegene/l systems," in *Proceedings of the 19th annual international conference on Supercomputing*. ACM, 2005, pp. 253–262.

[27] S. Kumar, G. Dozsa, G. Almasi, P. Heidelberger, D. Chen, M. E. Giampapa, M. Blocksome, A. Faraj, J. Parker, J. Ratterman, B. Smith, and C. J. Archer, "The deep computing messaging framework: Generalized scalable message passing on the blue gene/p supercomputer," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ser. ICS '08. New York, NY, USA: ACM, 2008, pp. 94–103. [Online]. Available: http://doi.acm.org/10.1145/1375527.1375544

[28] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, "Mpi collective communications on the blue gene/p supercomputer: Algorithms and optimizations," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*. IEEE, 2009, pp. 63–72.

[29] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski *et al.*, "The ibm blue gene/q compute chip," *Micro, IEEE*, vol. 32, no. 2, pp. 48–60, 2012.

[30] S. Kumar, A. Mamidala, P. Heidelberger, D. Chen, and D. Faraj, "Optimization of mpi collective operations on the ibm blue gene/q supercomputer," *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 4, pp. 450–464, Nov. 2014. [Online]. Available: http://dx.doi.org/10.1177/1094342014552086

[31] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li *et al.*, "The percs high-performance interconnect," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010, pp. 75–82.

[32] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "Netagg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 249–262.

[33] B. A. snf E. Froese snf L. Kaplan and D. Roweth, "Cray xc series network," Cray Inc., Tech. Rep., 2012.

[34] R. L. Graham, S. Poole, P. Shamis, G. Bloch, N. Bloch, H. Chapman, M. Kagan, A. Shahar, I. Rabinovitz, and G. Shainer, "Connectx-2 infiniband management queues: First investigation of the new support for network offloaded collective operations," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 53–62. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2010.9

[35] B. Barrett, R. Brightwell, S. Hemmert, K. Pedretti, K. Wheeler, K. D. Underwood, R. Reisen, A. B. Maccabe, and T. Hudson, "The portals 4.0 network programming interface, technical report sand201210087," Sandia National Laboratory, Tech. Rep., 2012.

[36] T. Schneider, T. Hoefler, R. Grant, B. Barrett, and R. Brightwell, "Protocols for Fully Offloaded Collective Operations on Accelerated Network Adapters," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, Oct. 2013, pp. 593–602.

[37] http://www.mellanox.com/page/products_dyn?product_family=189&mtag=hpc-x.

[38] http://www.mellanox.com/page/products_dyn?product_family=104&mtag=fca2.

[39] http://mvapich.cse.ohio-state.edu/overview/.

[40] http://mvapich.cse.ohio-state.edu/benchmarks/.

[41] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Comput. Phys.*, vol. 12, no. 6, pp. 620–631, Nov. 1998. [Online]. Available: http://dx.doi.org/10.1063/1.168744