DOI: 10.14529/jsfi170401

Towards A Data Centric System Architecture: SHARP

Gil Bloch[1], *Devendar Bureddy*[2], *Richard L. Graham*[3], *Gilad Shainer*[2], *Brian Smith*[3]

Increased system size and a greater reliance on utilizing system parallelism to achieve computational needs, requires innovative system architectures to meet the simulation challenges. The SHARP technology is a step towards a data-centric architecture, where data is manipulated throughout the system. This paper introduces a new SHARP optimization, and studies aspects that impact application performance in a data-centric environment. The use of UD-Multicast to distribute aggregation results is introduced, reducing the letency of an eight-byte *MPI_Allreduce()* across 128 nodes by 16%. Use of reduction trees that avoid the inter-socket bus further improves the eight-byte *MPI_Allreduce()* latency across 128 nodes, with 28 processes per node, by 18%. The distribution of latency across processes in the communicator is studied, as is the capacity of the system to process concurrent aggregation operations.

Keywords: data centric architecture, SHARP, collectives, MPI.

Introduction

The challenge of providing increasingly unprecedented levels of effective computing cycles, for tightly coupled computer-based simulations, continues to pose new technical hurdles. With each hurdle traversed, a new challenge comes to the forefront, with many architectural features emerging to address these problems. This has included the introduction of vector compute capabilities to single processor systems, such as the CDC Star-100 [28] and the Cray-1 [27], followed by the introduction of small-scale parallel vector computing, such as the Cray-XMP [5], custom-processor-based tightly-coupled MPPs, such as the CM-5 [21] and the Cray T3D [17], followed by systems of clustered commercial-off-the-shelf micro-processors, such as the Dell PowerEdge C8220 Stampede at TACC [30] and the Cray XK7 Titan computer at ORNL [24]. For a decade or so the latter systems relied mostly on Central Processing Unit (CPU) frequency upticks to provide the increase in computational power. But, as a consequence of the end of Dennard scaling [9], 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), [15], to keep scaling the system performance. 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.

With increasing compute engine counts, system architectures have continued to be CPU centric, with these system elements being involved in the vast majority of data manipulation. This has resulted in unnecessary data movement and undesirable competition between computational, communication, storage and other needs for the same computational resources. A Data-Centric system architecture, which co-locates computational resources and data throughout the system, enables data to be processed all across the system, and not only by CPU's

[1]Mellanox Technologies, Inc., Yokneam, Israel
[2]Mellanox Technologies, Inc., Sunnyvale, California, U.S.
[3]Mellanox Technologies, Inc., Knoxville, Tennessee, U.S.

at the edge. For example, data can be manipulated as it is being transferred within the data center network as part of a collective operation. This type of approach addresses latency and other performance bottlenecks that exist in the traditional CPU-Centric architecture. Mellanox focuses on CPU offload technologies designed to process data as it moves through the network, either by the Host Channel Adapter (HCA) 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.

InfiniBand technologies are being transformed to support such data-centric system architectures. These include technologies such as SHARP for handling data reduction and aggregation, hardware-based tag matching and Network data hardware-gather scatter capabilities. These technologies are used to process data and network errors at the network levels, without the need for data to reach a CPU, reducing overall volume of transferred data and system resilience.

This paper extends the investigation of the the SHARP technology previously introduced [12] for offloading aggregation and reduction operations to InfiniBand switches. The paper is organized as follows: Section 1 presents previous related work in offload technologies. Section 2 describes the new UD-Multicast protocol which utilizes multiple children in the reduction tree to avoid using internal node interconnect between sockets. Section 3 describes the benchmarks and applications investigated, and discusses the distribution of latencies across processes in a communicator and the network's ability to process multiple reduction operations concurrently. The final section provides a summary and discussion of the work presented.

1. Previous Work

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

Algorithmic work performed by Venkata et al. [33] 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 [31]. Vadhiar et al. [32] presented implementations of blocking reduction, gather and broadcast operations using sequential, chain, binary, binomial tree and Rabenseifner algorithms. Hoefler et al. [16] 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. [6] and [22], 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 [26] 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 [11] which uses torus interconnect [1], provided up to twice throughput performance gain of all-to-all collective operations [2, 20]. 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 [18]. MPI collectives optimization algorithms for this generation of Blue Gene were analyzed in [10]. The recent version Blue Gene/Q [14] provides additional performance improvements for MPI collectives [19]. On a 96,304 node system, the latency of a short allreduce is about 6.5 μ-seconds. IBM's PERCS system [4] fully offloads collective reduction operations to hardware. Finally, Mai et al. presented the NetAgg platform [23], which uses in-network middleboxes for partition/aggregation operations, to provide efficient network link utilization. Cray's Aries network [3] 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 APIs have been proposed for offloading collective operation management to the HCA. This includes the Mellanox's CORE-Direct [13], protocol, Portal 4.0 triggered operations [7], and an extension to Portals 4.0 [29]. 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.

2. 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 short vectors, and include barrier synchronization, and small data reductions. As previously mentioned, the SHARP protocol has already been described in detail, therefore, only a brief description is provided in this section, highlighting the new hardware capability that is introduced.

SHARP provides an abstraction describing data reduction and aggregation. The protocol defines aggregation nodes (ANs) which form the nodes of a reduction tree. These trees overlay a physical network. Figure 1 shows an example of a physical network topology, with Fig. 2 describing a possible reduction tree constructed over this physical topology. The aggregation nodes are colored in red, with the leaves of the tree, the blue stars, being source of the data.

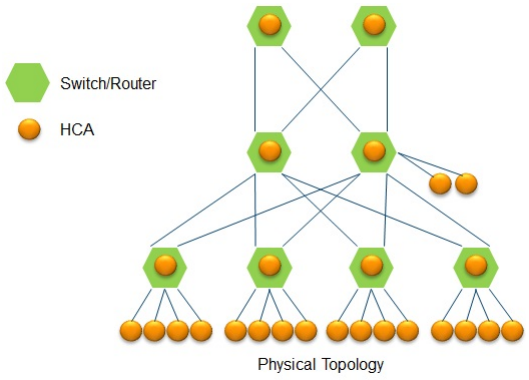


Figure 1. Physical Network Topology

Aggregation operations are defined for SHARP groups. These groups are formed as subtrees of SHARP trees, where multiple groups may be formed from a given SHARP tree. Figure 3 gives an example of a SHARP group of size eight.

An aggregation operation is performed with participation of each member of the aggregation group. To initiate such an operation, members of the aggregation group send their aggregation request message to their leaf aggregation node. The aggregation request header contains all needed information to perform the aggregation, and includes the data 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. The root aggregation node performs the final aggregation producing the result of the aggregation operation.

This aggregation result is distributed in up to two of several possible ways. 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. An aggregation tree can be used to distribute the data in these cases.

The new hardware capability described in this paper is that 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.

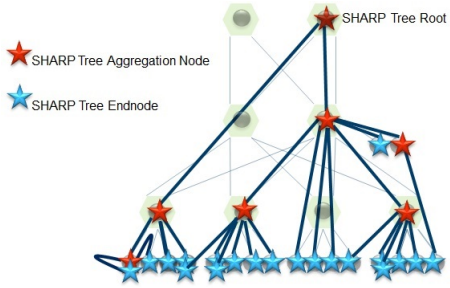


Figure 2. Logical SHARP Tree. Note that in the SHARP abstraction an Aggregation Node may be hosted by an end-node

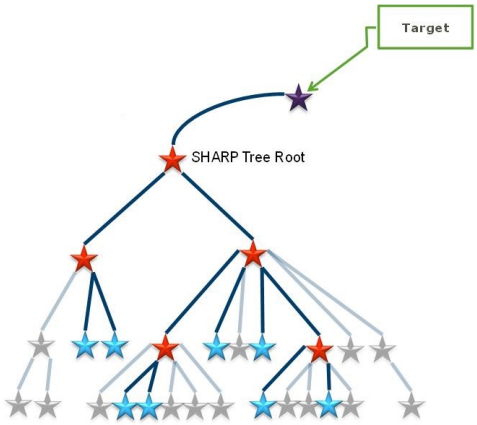


Figure 3. A SHARP group defined for the SHARP tree. The red stars designate AN's and the blue stars the tree leaves

The protocol does not define the data transport, so that communication between AN's 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 reordering, requiring a reliable transport which provides reliable in-order delivery of packets to the upper layer.

2.1. SwitchIB-2-Based Aggregation Support

In the SwitchIB-2 implementation, 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 Multicast group.

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.

Requests are collected in the TCA, with the reduction performed only after all operands are available, in a predetermined and fixed order. SwitchIB-2 implements a predictable operation ordering to enable repeatable results regardless of the order of arrival of the aggregation requests.

When using hardware multicast to distribute the aggregation results, the result also needs to be distributed with a reliable protocol to ensure delivery of these results.

3. Benchmark Results

To evaluate the SHARP capabilities, both low-level MPI benchmarks, as well as an application level benchmark are used.

A 128 host system is used for these experiments. Each node has two 14-core Broadwell CPUs running at 2.60 GHz, with 256GB of RAM memory. ConnectX-4 HCAs are used running at 100Gb/s. The fabric uses a two-level fat-tree with SwitchIB-2 switches and eight leaf switches, each connecting to 16 hosts. The hosts run RedHat Linux 7.2, and the tests were carried out with OFED 3.4-2.1.9.0. A pre-release version of HPC-X, the Mellanox supported MPI, is used, which includes a set of MPI collective routines that access and use the SHARP hardware capabilities, embedded in the SwitchIB-2 switches, to optimize the performance of the corresponding MPI collectives.

3.1. MPI-Level SHARP Measurements

The OSU *MPI_Allreduce()* test [25] is used to measure the SHARP latency.

Figure 4 shows the latency of *MPI_Allreduce()* operations as a function of message size and the mode of result distribution, with one process per-node. Using UD multicast for distributing the result takes advantage of the O(1) multicast capabilities for improved performance, but is unreliable (bit error rate being on the order of 10^{-15}) requiring the additional RC result distribution to provide the result when a UD packet is dropped. Using UD multicast and RC to distribute the results improves latency in the range of 15-58% relative to using RC only for

this distribution, even with the duplicate result distribution. The improvement relative to the host-based approach is in the range of 143 to 385 percent.

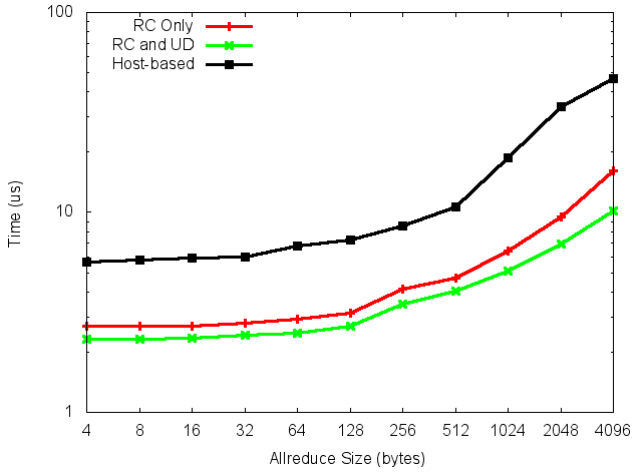


Figure 4. 128 node *MPI_Allreduce()* average latency with different modes of result distribution. A comparison to the host-only algorithm is also included. Latency is reported in μ-seconds

SHARP reduction trees assume some sort of host-level aggregation prior to sending data to the leaf AN, because of the limitation on AN's radix. Figure 5 shows the latency of the *MPI_Allreduce()* operation when using one connection per socket (2 channels) into the SHARP reduction tree, avoiding reduction over the internal chip network, and one connection per node (1 channel). As the results show, for messages up to 1024 bytes in size, this reduces latency by more than ten percent. With larger messages, an increase in latency is observed. The two-channel case eliminates the host-side intra-socket reduction steps, it increases the leaf AN radix by a factor of two. As the vector length increases, this manifests itself with a larger latency relative to the one-channel case.

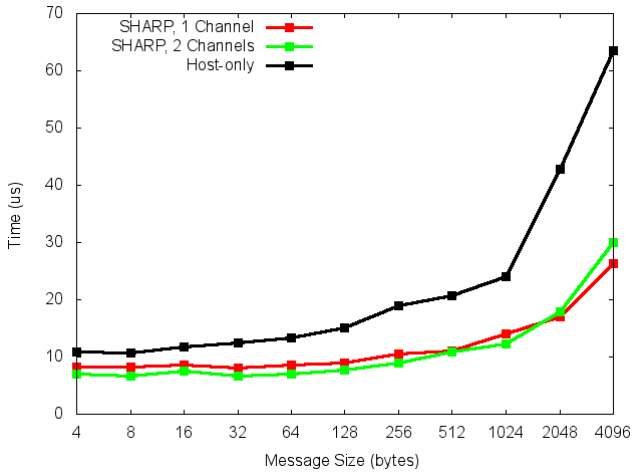


Figure 5. 128 node, 28 processes per-node *MPI_Allreduce()* average latency in μ-seconds

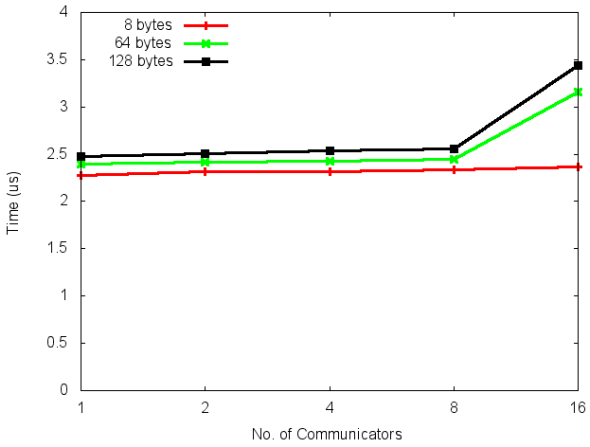
To get a better understanding of the spread in completion times across the communicator, several metrics are collected to characterize this behavior. Table 1 lists the average *MPI_Allreduce()* latencies, along with quartile data, minimum value and maximum value to describe the data distribution, using UD-multicast for result distribution, and one process per node. These are reported for the average of the full collective operation (measured as as the average of the

collective operation) and for the the completion of each of the individual ranks in the communicator. As expected, there is greater variance in individual completion times, as compared with the average per-collective completion time. Also, we see that the SHARP based collectives have a much smaller per-rank latency range.

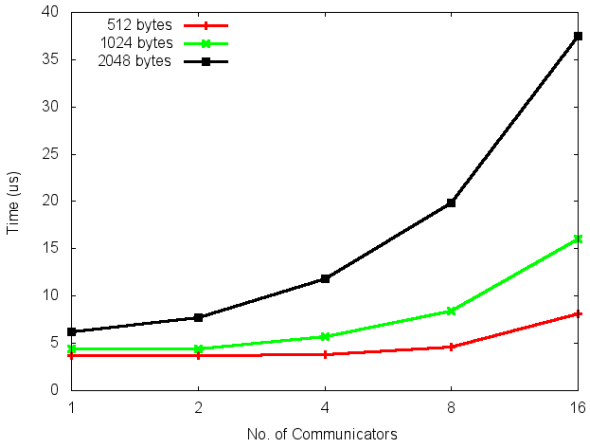
Table 1. *MPI_Allreduce()* Latency (μsec) Distribution of a 127 Node Cluster with One Process Per Node

	Size (B)	Ave.	Quartiles min, Q1, Q2, Q3, max Per Operation Average	Quartiles min, Q1, Q2, Q3, max Per Process Data
SHARP	4	2.39	2.38, 2.39, 2.39, 2.40, 2.41	2.16, 2.35, 2.37, 2.41, 13.49
Host		5.09	4.99, 5.07, 5.09, 5.10, 5.13	2.34, 4.74, 4.88, 5.13, 12.69
SHARP	8	2.39	2.37, 2.38, 2.39, 2.39, 2.40	2.14, 2.35, 2.37, 2.41, 2.78
Host		5.18	5.11, 5.18, 5.18, 5.19, 5.21	2.51, 4.83, 4.98, 5.20, 15.49
SHARP	16	2.41	2.40, 2.40, 2.41, 2.41, 2.42	2.20, 2.37, 2.39, 2.43, 2.90
Host		5.26	5.20, 5.25, 5.26, 5.28, 5.31	2.48, 4.91, 5.05, 5.27, 16.15
SHARP	32	2.47	2.47, 2.48, 2.48, 2.48, 2.49	2.26, 2.45, 2.47, 2.50, 3.02
Host		5.32	5.26, 5.31, 5.32, 5.33, 5.38	2.48, 4.97, 5.11, 5.36, 13.64
SHARP	64	2.55	2.55, 2.55, 2.56, 2.56, 2.57	2.26, 2.52, 2.55, 2.57, 3.00
Host		5.98	5.72, 5.98, 6.00, 6.00, 6.04	2.70, 5.65, 5.80, 6.01, 18.38
SHARP	128	2.76	2.75, 2.76, 2.76, 2.76, 2.77	2.44, 2.72, 2.74, 2.77, 10.30
Host		6.43	6.17, 6.43, 6.44, 6.45, 6.51	3.23, 6.10, 6.27, 6.50, 13.66
SHARP	256	3.52	3.51, 3.51, 3.52, 3.52, 3.53	3.04, 3.48, 3.51, 3.54, 7.37
Host		7.55	7.38, 7.54, 7.57, 7.58, 7.62	4.19, 7.29, 7.42, 7.63, 16.36
SHARP	512	4.10	4.07, 4.07, 4.07, 4.10, 4.25	3.63, 4.05, 4.08, 4.14, 10.70
Host		9.16	8.96, 9.14, 9.17, 9.19, 9.22	4.05, 8.93, 9.04, 9.21, 24.66
SHARP	1024	5.19	5.11, 5.15, 5.18, 5.21, 5.32	4.68, 5.07, 5.15, 5.28, 7.70
Host		18.49	16.24, 17.36, 18.27, 19.60, 20.52	11.38, 17.33, 18.67, 19.52, 31.69
SHARP	2048	7.55	7.52, 7.54, 7.55, 7.56, 7.58	5.61, 7.22, 7.51, 7.80, 16.65
Host		33.47	31.33, 32.56, 33.60,3 4.27, 36.89	28.83, 32.48, 33.49, 34.25, 50.40
SHARP	4096	12.34	12.30, 12.33, 12.34, 12.35, 12.39	10.38, 11.54, 11.99, 13.06, 17.27
Host		45.99	42.60, 45.10, 46.06, 46.80, 49.49	39.15, 44.99, 45.95, 46.89, 58.68

For the SHARP capabilities to be useful in a general purpose production system, where multiple jobs run concurrently, potentially sharing ANs, it is useful to study the systems ability to support concurrent SHARP operations. The system's capacity to service concurrent collective operations is studied by running multiple collective operations at the same time, using completely overlapping SHARP-tree groups. The OSU-latency test was modified to run concurrent collective operations with non-overlapping MPI Communicators, with the MPI process layout configured to achieve this overlap. As the results show in Fig. 6 for communicators of size eight, SHARP is able to accommodate many outstanding operations very well. Latency starts to degrade at a message size of 2048 bytes, with eight concurrent operations, where as many as sixty four operations are in flight. With sixteen concurrent operations, latency is impacted by about 30% with a message size of 64 bytes.



(a) Small-size Allreduce



(b) Large-size Allreduce

Figure 6. SHARP *MPI_Allreduce()* latency (in μ-seconds) for 128 nodes with varying simultaneous communicators

Table 2 presents the *MPI_Allreduce()* latency as a function of the number of outstanding SHARP operations each group is configured to allow. The eight byte data requires only one SHARP-level operation per MPI operation, whereas the 2048 byte reduction requires eight such operations. As expected, we see that the eight byte reduction is minimally impacted by the number of allowed outstanding SHARP operations, except in the eight communicator test, where there are insufficient resources for all communicators, and the test does not run as written. The 2048 byte MPI-level operation is negatively impacted by the lack of sufficient resources to pipeline the entire operation at once, but even with only two outstanding SHARP operations supported, there is the benefit of some pipelining, with the latency being less than four times that of the eight operation case.

3.2. Application Benchmarks

Table 3 shows the result of running the Algebraic Multi-Grid (AMG) [8] micro benchmark on 64 nodes, with 28 processes per node. The AMG benchmark uses an eight byte data reduction. On average, running five of the AMG test cases (Laplace, 27 point, Jumps, def/pool1 and def/pool0) an average improvement of 1.8% in total test run time was measured when using

Table 2. Eight Process *MPI_Allreduce()* Average Latency (μsec) as a Function of the Number of Communicators Operating in Parallel and as a Function of Maximum Outstanding SHARP Operations (OSOs) Available

8 bytes Total OSOs	OSOs per Comm	1 Comm	2 Comm	4 Comm	8 Comm
8	1	2.24	2.263	2.260	N/A
16	1	2.210	2.253	2.260	2.238
32	2	2.210	2.245	2.250	2.236
2048 bytes OSOs	**OSOs per Comm**	**1 Comm**	**2 Comm**	**4 Comm**	**8 Comm**
32	2	11.890	11.990	12.154	14.991
64	4	7.695	7.783	8.374	12.229
128	8	5.495	5.533	6.710	10.351

SHARP. The figure of merit used is system-size * number-of-iterations / (solve-time in units of seconds).

Table 3. AMG Figure-of-Merit (Higher is Better) Data for Five Different Tests, Run on 64 Nodes with 28 Processes Per Node, and a System Configured with Low System Noise

Job Type	Laplace	27pt	Jumps	Pooldist_1	Pooldist_0
non-SHARP	2.10E+09	1.64E+09	2.84E+09	2.45E+09	3.82E+09
SHARP	2.21E+09	1.68E+09	2.86E+09	2.40E+09	3.92E+09
% Change	5.2	2.4	0.7	-2.0	2.6

Discussion and Conclusions

To improve MPI-level aggregation performance, UD-Multicast is used to distribute results from the root of the aggregation-tree, and SHARP trees that avoid using the host's inter-socket bus for aggregations are employed. Employing UD-multicast to distribute the aggregated values reduces overall operation latency, even though the result is sent twice to ensure reliable delivery, once using UD-Multicast and once with RC. The UD packets allow for fast result distribution, with a very low packet-rate loss. The RC packets sent to ensure data delivery arrive a little later, and impact latency only when the UD packets are lost. This improves eight byte reduction at 128 nodes by 16%, and the 4096 byte latency by 58%. The distribution using UD-Multicast benefits from the switch's ability to replicate the data packet to all ports relevant to the multicast group in parallel, whereas the RC packet replication has some degree of serialization. In addition, for small message distribution message rate, rather than bandwidth, is the primary performance limiter. The high message rate, of 195 messages per port, per μ-second supported by the SwitchIB-2 device, is capable of handling the duplicated data.

Using the intra-host bus for data exchange between sockets can be expensive relative to the intra-socket communication. It is frequently more efficient to avoid using this bus when accessing the network, and therefore a similar approach has been investigated for SHARP reductions. As the results show, aggregating data on a per-socket basis also helps reduce operation latency for small message sizes, reducing the eight byte operation latency by 16% at 128 nodes. However, as the data size increases, competition for the PCIe bus bandwidth from the host to the network and the two fold increase in AN radix at the leaf switches make this particular optimization undesirable.

In general purpose data-centric environments, with multiple jobs running on the system at the same time jobs compete for a fixed set of resources, unless special care has been taken to isolate the resources used by separate jobs. In the case of SHARP the AN resources are an additional set of resources, beyond the host and other network resources that may be shared. The impact of such sharing on the SHARP latencies has been studied by running concurrent reductions on the same reduction-tree and limiting the number of concurrent aggregations.

To study the effectiveness of the protocol in a multi-job scenario, where some ANs may be used by multiple jobs, we ran up to sixteen concurrent collectives simultaneously. This is expected to be a worse-case type of scenario, because the test forces the collective operation concurrency. Since application runs typically are not synchronized, and they do more than just run collective operations, the impact on concurrent running applications using the same AN resources is expected to be less. The results show that the impact on the small message reduction latency is small, but as the message size increases the impact of this sharing becomes noticeable due to the competition for bandwidth. At 2048 byte message size and 128 nodes, a small impact is noticed when two operations are running concurrently, but with four it is still advantageous to use the SHARP protocol over the host-based protocol.

We also observed that when there are insufficient resources to pipeline a reduction operation with independent resources, there are still benefits to such optimization when compared with the host-based approach. A 2048 byte message size and 128 nodes requires eight OSOs for the full message reduction to be concurrently in flight. However, providing only two such OSOs still reduces the operation latency relative to the host-based approach.

Finally, collective operations are known to amplify application load imbalance. Looking at the per-process spread in collective operations, we see that the SHARP based collectives are less susceptible to imbalance within the collective algorithms themselves, thus supporting application scalability better than the host-based algorithms.

In conclusion, this paper has introduced the ability to use UD-multicast for aggregation result distribution and presented several aspects of the SHARP protocol not previously examined. Benchmark and application results show that the protocol is effective, and help to show how to best utilize the underlying SHARP capabilities in a general purpose data-centric environment.

References

1. Adiga, N.R., Blumrich, M.A., Chen, D., Coteus, P., et al.: Blue Gene/L torus interconnection network. IBM Journal of Research and Development 49(2/3), 265 (2005), DOI: 10.1147/rd.492.0265

2. Almási, G., Heidelberger, P., Archer, C.J., Martorell, X., Erway, C.C., Moreira, J.E., Steinmacher-Burow, B., Zheng, Y.: Optimization of MPI collective communication on BlueGene/L systems. In: Proceedings of the 19th annual international conference on Supercomputing. pp. 253–262. ACM (2005), DOI: 10.1145/1088149.1088183

3. Alverson, B., Froese, E., Kaplan, L., Roweth, D.: Cray XC series network. Tech. rep., Cray Inc. (2012), https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf, accessed: 2017-10-01

4. Arimilli, B., Arimilli, R., Chung, V., Clark, S., Denzel, W., Drerup, B., Hoefler, T., Joyner, J., Lewis, J., Li, J., et al.: The PERCS high-performance interconnect. In: High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on. pp. 75–82. IEEE (2010), DOI: 10.1109/HOTI.2010.16

5. August, M.C., Brost, G.M., Hsiung, C.C., Schiffleger, A.J.: Cray X-MP: The birth of a supercomputer. Computer 22(1), 45–52 (1989), DOI: 10.1109/2.19822

6. Barnett, M., Littlefield, R.J., Payne, D.G., van de Geijn, R.A.: Global combine on mesh architectures with wormhole routing. In: The Seventh International Parallel Processing Symposium, Proceedings, Newport Beach, California, USA, April 13-16, 1993. pp. 156–162 (1993), DOI: 10.1109/IPPS.1993.262873

7. Barrett, B., Brightwell, R., Hemmert, S., Pedretti, K., Wheeler, K., Underwood, K.D., Reisen, R., Maccabe, A.B., Hudson, T.: The Portals 4.0 network programming interface, technical report SAND201210087. https://www.osti.gov/scitech/biblio/1088065 (2012), accessed: 2017-10-01

8. CORAL Collaboration: Benchmark codes. https://asc.llnl.gov/CORAL-benchmarks/#amg2013, accessed: 2017-10-01

9. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: Computer Architecture (ISCA), 2011 38th Annual International Symposium on. pp. 365–376. IEEE (2011), DOI: 10.1145/2000064.2000108

10. Faraj, A., Kumar, S., Smith, B., Mamidala, A., Gunnels, J.: MPI collective communications on the Blue Gene/P Supercomputer: Algorithms and optimizations. In: High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on. pp. 63–72. IEEE (2009), DOI: 10.1109/HOTI.2009.12

11. Gara, A., Blumrich, M.A., Chen, D., Chiu, G.T., Coteus, P., Giampapa, M.E., Haring, R.A., Heidelberger, P., Hoenicke, D., Kopcsay, G.V., et al.: Overview of the Blue Gene/L system architecture. IBM Journal of Research and Development 49(2), 195–212 (2005), DOI: 10.1147/rd.492.0195

12. Graham, R.L., Bureddy, D., Lui, P., Rosenstock, H., Shainer, G., Bloch, G., Goldenerg, D., Dubman, M., Kotchubievsky, S., Koushnir, V., Levi, L., Margolin, A., Ronen, T., Shpiner, A., Wertheim, O., Zahavi, E.: Scalable hierarchical aggregation protocol (SHArP): A hardware architecture for efficient data reduction. In: Proceedings of the First Workshop on Optimization of Communication in HPC. pp. 1–10. COM-HPC '16, IEEE Press, Piscataway, NJ, USA (2016), DOI: 10.1109/COM-HPC.2016.6

13. Graham, R.L., Poole, S., Shamis, P., Bloch, G., Bloch, N., Chapman, H., Kagan, M., Shahar, A., Rabinovitz, I., Shainer, G.: 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. pp. 53–62. CCGRID '10, IEEE Computer Society, Washington, DC, USA (2010), DOI: 10.1109/CCGRID.2010.9

14. Haring, R.A., Ohmacht, M., Fox, T.W., Gschwind, M.K., Satterfield, D.L., Sugavanam, K., Coteus, P.W., Heidelberger, P., Blumrich, M.A., Wisniewski, R.W., et al.: The IBM Blue Gene/Q compute chip. Micro, IEEE 32(2), 48–60 (2012), DOI: 10.1109/MM.2011.108

15. Herbordt, M.C., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., DiSabello, D.: Achieving high performance with FPGA-based computing. Computer 40(3), 50 (2007), DOI: 10.1109/MC.2007.79

16. Hoefler, T., Squyres, J.M., Rehm, W., Lumsdaine, A.: A case for nonblocking collective operations. In: In Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops. pp. 155–164. Springer (2006), DOI: 10.1007/11942634_17

17. Kessler, R., Schwarzmeier, J.: Cray T3D: a new dimension for Cray Research. In: Compcon Spring '93, Digest of Papers. pp. 176 –182 (1993), DOI: 10.1109/CMPCON.1993.289660, accessed: 2017-12-19

18. Kumar, S., Dozsa, G., Almasi, G., Heidelberger, P., Chen, D., Giampapa, M.E., Blocksome, M., Faraj, A., Parker, J., Ratterman, J., Smith, B., Archer, C.J.: 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. pp. 94–103. ICS '08, ACM, New York, NY, USA (2008), DOI: 10.1145/1375527.1375544

19. Kumar, S., Mamidala, A., Heidelberger, P., Chen, D., Faraj, D.: Optimization of MPI collective operations on the IBM Blue Gene/Q Supercomputer. Int. J. High Perform. Comput. Appl. 28(4), 450–464 (2014), DOI: 10.1177/1094342014552086

20. Kumar, S., Sabharwal, Y., Garg, R., Heidelberger, P.: Optimization of all-to-all communication on the Blue Gene/L Supercomputer. In: Proceedings of the 2008 37th International Conference on Parallel Processing. pp. 320–329. ICPP '08, IEEE Computer Society, Washington, DC, USA (2008), DOI: 10.1109/ICPP.2008.83

21. Leiserson, C.E., Abuhamdeh, Z.S., Douglas, D.C., et al.: The network architecture of the connection machine CM-5 (extended abstract). In: Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures. pp. 272–285. SPAA '92, ACM, New York, NY, USA (1992), DOI: 10.1145/140901.141883

22. Liu, V.W., Chen, C., Chen, R.B.: Optimal all-to-all personalized exchange in d-nary banyan multistage interconnection networks. Journal of Combinatorial Optimization 14(2), 131–142 (2007), DOI: 10.1007/s10878-007-9065-5

23. Mai, L., Rupprecht, L., Alim, A., Costa, P., Migliavacca, M., Pietzuch, P., Wolf, A.L.: 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. pp. 249–262. ACM (2014), DOI: 10.1145/2674005.2674996

24. Oak Ridge National Laboratory Leadership Computing Facility: Titan Cray XK7. https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/, accessed: 2017-10-01

25. Ohio State University Network-Based Computing Laboratory: OSU microbenchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/, accessed: 2017-10-01

26. Petrini, F., Coll, S., Frachtemberg, E., Hoisie, A.: Hardware-and-software-based collective communication on the Quadrics network. (2001), http://www.osti.gov/scitech/servlets/purl/975699, accessed: 2017-10-19

27. Russell, R.M.: The CRAY-1 computer system. Commun. ACM 21(1), 63–72 (1978), DOI: 10.1145/359327.359336

28. Schneck, P.B.: Supercomputer Architecture, chap. The CDC STAR-100, pp. 99–117. Springer US, Boston, MA (1987), DOI: 10.1007/978-1-4615-7957-1_5

29. Schneider, T., Hoefler, T., Grant, R., Barrett, B., Brightwell, R.: Protocols for Fully Offloaded Collective Operations on Accelerated Network Adapters. In: Parallel Processing (ICPP), 2013 42nd International Conference on. pp. 593–602 (2013), DOI: 10.1109/ICPP.2013.73

30. Texas Advanced Computing Center: Stampede Supercomputer. https://www.tacc.utexas.edu/systems/stampede, accessed: 2017-12-19

31. Thakur, R., Rabenseifner, R.: Optimization of collective communication operations in MPICH. International Journal of High Performance Computing Applications 19, 49–66 (2005), DOI: 10.1177/1094342005051521

32. Vadhiyar, S.S., Fagg, G.E., Dongarra, J.: Automatically tuned collective communications. In: In Proceedings of SC99: High Performance Networking and Computing. p. 3. IEEE Computer Society (2000), DOI: 10.1109/SC.2000.10024

33. Venkata, M.G., Shamis, P., Sampath, R., Graham, R.L., Ladd, J.S.: Optimizing blocking and nonblocking reduction operations for multicore systems: Hierarchical design and implementation. In: Cluster Computing (CLUSTER), 2013 IEEE International Conference on. pp. 1–8. IEEE (2013), DOI: 10.1109/CLUSTER.2013.6702676