Towards A Data Centric System Architecture: SHARP

Richard L. Graham[1]***, Gilad Shainer***[1]

Keywords: Data Centric Architecture, SHARP, Collectives.

Long Abstract

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, followed by the introduction of small-scale parallel vector computing, custom-processor-based tightly-coupled MPPs, followed by systems of clustered commercial-off-the-shelf micro-processors. 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 [1], 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), to keep scaling the system performance.

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 and communication needs for the same system resources. A Data-Centric architecture enables data to be analyzed everywhere, and in particular as data is being transferred within the data center network. This approach solves the latency and other performance bottlenecks that exist in the traditional CPU-Centric architecture. Mellanox solutions for the Data-Centric architecture include both hardware and software elements to meet the needs of the next generation of extreme scale applications.

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 transfered data and system resilience.

This paper focuses on the SHARP technology previously introduced [2]. This is used to define aggregation trees within the network, with interior nodes mapped to switches, and leaf nodes to system nodes. Reduction groups are implemented as subtrees of the aggregation trees. Collective operations are mapped to these subtrees, with the data originating at the hosts and reduced as it makes its way up the tree to the root using the RC transport to send the data between nodes in the tree. The result is sent from the root down the tree to the hosts using the RC transport to send data between the tree nodes. This paper introduces the use of UD-

[1]Mellanox Technologies, Inc.

Multicast for result distribution, the use of SHARP reduction trees to avoid using the CPU's inter-socket bus for data reductions, as well as a detailed study of its performance characteristics. The OSU latency benchmark which uses the MPI_Allreduce() algorithm is used to look at the SHARP performance characteristics in isolation . The impact of the SHARP capabilities on the CORAL Adaptive Multi-Grid linear solver benchmark is also presented.

Table 1 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 and RC down the tree improves the latency in the range of 5-15%. Using UD multicast for distributing the result is to take 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.

size	RC Only	RC and UD (Improvement)	Host based
4	2.69	2.35 (14.5%)	5.68
8	2.69	2.35 (14.5%)	5.78
16	2.70	2.36 (14.4%)	5.89
32	2.77	2.44 (13.5%)	6.00
64	2.92	2.52 (15.9%)	6.78
128	3.14	2.72 (15.4%)	7.3
256	3.42	3.98 (-16.4%)	8.59
512	3.91	3.55 (10.1%)	10.69
1024	5.52	4.74 (16.4%)	18.67
2048	8.61	7.54 (14.2%)	33.7
4096	14.62	13.88 (5.3%)	46.89

Table 1. 128 node *MPI_Allreduce()* average latency (μsec).

Table 2 shows the latency of the *MPI_Allreduce()* operation when using one connection per socket into the SHARP reduction tree, avoiding reduction over the internal chip network, and one connection per node. 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.

Table 3 lists the average *MPI_Allreduce()* latencies, along with standard deviation and quartile data 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.

The system's capacity to service concurrent collective operations is studied by running multiple collective operations at the same time, using completely overlapping SHRAP-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 table 4 for communicators of size eight, SHARP is able to accommodate well many outstanding operations. Latency starts to degrade at message size 2048 bytes, with eight concurrent operations, where as many as sixty four operations are in flight.

size (B)	SHARP 1 Channel	SHARP 2 channel (Improvement)	Host Based
4	8.19	6.93 (15%)	10.95
8	8.16	6.68 (18%)	10.68
16	8.65	7.52 (13%)	11.67
32	8.11	6.71 (17%)	12.52
64	8.66	7.05 (18%)	13.39
128	8.98	7.67 (14%)	15.07
256	10.44	9.01 (14%)	18.89
512	11.04	10.83 (1.9%)	20.72
1024	14.01	12.2 (13%)	24.11
2048	17.06	17.86 (-4.7%)	42.82
4096	26.23	29.92 (-14%)	63.44

Table 2. 128 node *MPI_Allreduce()* average latency (μsec).

	size (B)	Average	Quartiles min, Q1, Q2, Q3, max 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

Table 3. 127 node *MPI_Allreduce()* average latency (μsec).

The AMG benchmark uses an eight bytel data reduction. On average, running five of the AMG test cases (Laplace, 27 point, Jumps, def/pool1 and def/pool0) an average improvement

	size (B)	1 Comm	2 Comm	4 Comm	8 Comm
SHARP	4	2.37	2.40	2.41	2.44
Host		2.21	2.35	2.23	2.22
SHARP	8	2.38	2.40	2.41	2.45
Host		2.22	2.35	2.24	2.35
SHARP	16	2.39	2.43	2.43	2.46
Host		2.26	3.38	2.28	2.36
SHARP	32	2.43	2.45	2.46	2.48
Host		2.29	3.42	2.31	2.40
SHARP	64	2.47	2.49	2.51	2.53
Host		2.75	3.87	2.76	2.84
SHARP	128	2.57	2.60	2.62	2.64
Host		2.90	2.99	2.90	2.98
SHARP	256	3.12	3.15	3.19	3.22
Host		3.15	3.27	3.17	3.26
SHARP	512	3.71	3.72	3.77	3.04
Host		3.60	3.69	3.63	3.70
SHARP	1024	4.41	4.44	4.49	4.68
Host		7.84	7.89	7.84	7.80
SHARP	2048	5.68	5.76	5.83	8.31
Host		14.31	14.40	14.32	14.22
SHARP	4096	9.34	9.34	9.10	
Host		19.05	19.16	19.18	

Table 4. 8 node *MPI_Allreduce()* average latency (μsec).

of 1.8% improvement in total test run time was measured. These tests were run on 64 nodes, with 28 processes per node.

References

1. Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

2. Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction. In *Proceedings of the First Workshop on Optimization of Communication in HPC*, COM-HPC '16, pages 1–10, Piscataway, NJ, USA, 2016. IEEE Press.