Networks for High-Performance Computing

Louis Jenkins
University of Rochester

***Abstract*— In high-performance computing (HPC), low-latency and high-bandwidth networks are required to maintain high-performance in a distributed computing environment. According to the Top500, the two most widely-used networks in the top 10 supercomputers are Cray's Aries network interconnect, and Mellanox's Infiniband network interconnect, which both take radically different approaches towards their design. This survey explores the design of each network interconnect, the ways they provide reliability and fault-tolerance, their support for remote direct memory access (RDMA), and analyze the performance of an Infiniband cluster and a Cray-XC50 supercomputer. Finally, the near-future of high-performance interconnect, obtained from an interview with Steve Scott, Chief Technology Officer (CTO) of Cray Inc., a Hewlett Packard Enterprise company, is also examined.**

I. INTRODUCTION

To address the ever increasing needs for computational power, such computations have become more and more decentralized over time. The technological advancements from uni-processor to multi-processor systems, from single-socket to multi-socket NUMA domains, and from shared-memory to distributed memory, all are testaments to this fact. Decentralizing computations has many advantages, such as increasing of available *processing elements (PE)* available for balancing the computational workload across, but it does not come without its challenges. The lower down the memory hierarchy you go, the higher the latency, where sometimes the difference between one part of the hierarchy and another is as much as an order of magnitude. While in accessing memory that is *local* to the PE may have a lot of variability, such as the case of L1 Cache, L2 Cache, L3 Cache, and main memory (DRAM), they are usually within the span of nanoseconds, while accesses to memory that is *remote* to the PE can be in the span of microseconds to milliseconds. Computations that span across multiple PEs require communication over some type of network, and in the context of high-performance computing, the larger the bandwidth and smaller the latency, the better.

Supercomputers and compute clusters primarily tend to use one of two high-performance network interconnects, being Infiniband[1], [2] from Mellanox and Aries[3], successor of Gemini[4], [5], from Cray. Commodity clusters also require high-performance networks, although Infiniband may be excessive for their computational needs, and so they may opt for RDMA over Converged Ethernet (RoCE)[6], which is common in big data centers. The high-performance interconnects require their own unique design and implementation that minimizes latency, maximizes bandwidth, and ensures reliability of data sent between PEs on the network. These interconnects implement *Remote Direct Memory Access (RDMA)* which allows for accessing memory on a remote PE directly through the Network Interface Controller (NIC) without the intervention of the CPU, and/or even the GPU. RDMA allows for the the high-bandwidth and low-latency remote memory operations, both incoming and outgoing, to not saturate the CPU, freeing it up to perform more computations. RDMA for GPUs, such as NVidia's GPUDirect, allows for not just host-to-remote-device, but even device-to-remote-device transfers to perform without any unnecessary copying to and from the CPU, where normally transferring device-to-device would require device-to-host, host-to-host, and then a host-to-device. As well, RDMA can be used to *atomically* update remote memory such that it appears to happen all-at-once or not-at-all, which opens up many potential applications.

In section 2, the properties of a high-performance network are explored in detail, and in section 3, the design and implementation of Infiniband is examined, while in section 4, the design and implementation of Gemini and it's successor Aries are looked at in detail. In section 5, performance benchmark results from the Ohio State University (OSU) benchmarks are analyzed between a Cray-XC50 and Infiniband compute cluster, and in section 6, we take a look at the near future of high-performance computing, courteousy of Steve Scott, CTO of Cray Inc., a Hewlett Packard Enterprise company. Finally, in section 7, we have our conclusion.

II. PROPERTIES OF HIGH-PERFORMANCE NETWORKS

There are many characteristics and properties that make up networks suitable for high-performance computing. The structure of the network, that is it's topology, as well as the properties relating to fault tolerance and reliability, and the types of supported communication between PEs in the same network, are all important. A supercomputer or compute cluster lacking in any of these will be lacking as a whole, as each individual part plays a role in extracting as much performance possible from the underlying hardware.

A. Network Topology

The topology of a network describes the overall makeup and layout of switches, nodes, and links, and the ways they are connected. This layout is crucial for maximizing bandwidth, minimizing latency, ensuring reliability, and enabling fault tolerance in a network. There are many types of topological structures in a network, but only the most commonly used in the top supercomputers and compute

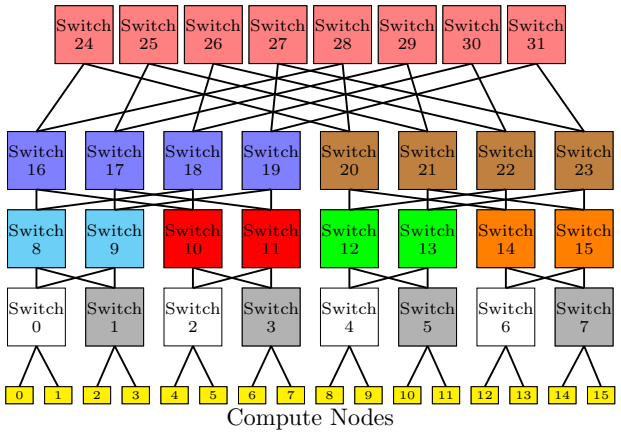

Fig. 1. Fat Tree Network Topology

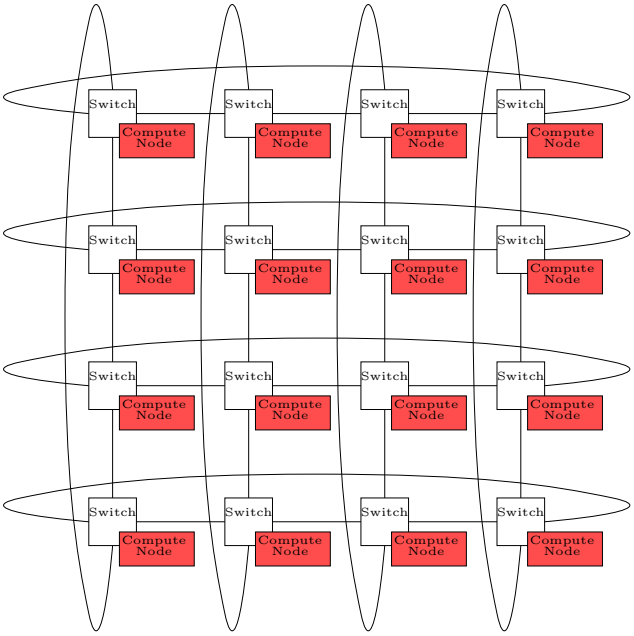

Fig. 2. 2D Torus Network Topology

clusters will be considered, and all else are outside the scope of this work.

The *Fat Tree* is a tree data structure where branches are larger, or "fatter", the farther up the tree they are. The 'thickness' of the branch is an analog for the amount of bandwidth available to that node; the fatter the branches, the larger the bandwidth. The fat tree, in the context of networking, have all non-leaf nodes in the tree as switches, with leaf nodes being PEs, as demonstrated in figure 1.[7] The Fat Tree is extremely cost effective, as adding a new PE to the network can be as simple as adding a logarithmic number of switches in the worst case. Communication from one PE to another is handled by *single source shortest path (SSSP)* routing, which sends packets up and down the tree from the source PE to the target PE.

The *Torus* topology, with a 2D Torus network shown in

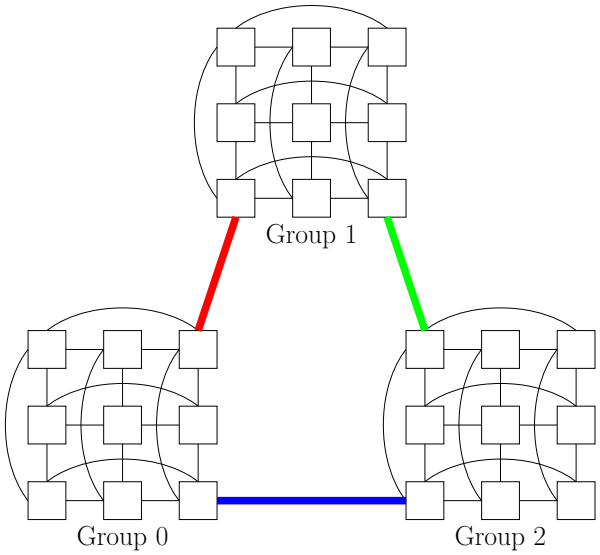

Fig. 3. Dragonfly Network Topology

Figure 2[7], can be generalized to N dimensions, where each the PEs are placed on a coordinate system. For example, in a 3D Torus network topology, each PE can connect to 6 different PEs in the $\pm x$ direction, $\pm y$ or in the $\pm z$ direction. The Torus offers a lot of diversity in routing, as there are a wide variety of ways that you can send data from a source PE to a target PE, and hence increased bandwidth, but comes at the downside of a higher cost and much more difficult setup.

The *Dragonfly* topology is a hybrid, in that PEs are grouped together, where at least one PE in a group is linked to some other PE in some other group. The topology used inside of the group is implementation dependent, but usually is the Fat Tree topology or a Torus or Mesh topology is used, as shown in Figure 3.

B. Fault Tolerance and Reliability

In a network, *reliability* is an important feature for ensuring consistency, and when done well, can serve to extract even greater performance. A packet in an unreliable network may be lost, or have to be retransmitted at the cost of additional time, or at absolute worst, could result in the acceptance of corrupted data that throws the application in jeopardy. In a strive to balance both reliability and performance, each high-performance network must make some rather unique design decisions and trade-offs that may be unique to how they are handled in your run-of-the-mill network. One way that packets are protected is via a *Cyclic Redundancy Check (CRC)*, which is adds a fixed-width checksum to the original payload, where the checksum is calculated in a way that is sensitive to the position and value of each individual bit. The usage of *Error Correcting Codes (ECC)* is also used in the link itself to add an additional layer of reliability and recoverability on an error.

When a PE goes down in a network, network traffic must be routed around it, as it may no longer be able to properly receive, send, or even forward packets that route through

Atomic Operation	Gemini	uGNI	RoCE	Infiniband
Add (Integer)	64	32,64	N/A	N/A
Add (Float)	N/A	32	N/A	N/A
Fetch-Add	64	32,64	64	64
And	64	32,64	N/A	N/A
Fetch-And	64	32,64	64	64
Or	64	32,64	N/A	N/A
Fetch-Or	64	32,64	64	64
Min (Integer)	64	32,64	N/A	N/A
Min (Float)	N/A	32	N/A	N/A
Max (Integer)	64	32,64	N/A	N/A
Max (Float)	N/A	32	N/A	N/A

TABLE I
ATOMIC MEMORY OPERATORS

it. Most networks have what is known as *adaptive routing*, where the route a packet travels is computed on-the-fly and takes into account cases where you have PE-failure in a network. The ability to route around a down PE is generally up to the topology of the network; a network without multiple paths may not be able to tolerate even a single PE failure, while a more diverse network may be able to handle a rather large number before the network goes down. While *single-source shortest path (SSSP)* may be used to optimize these routes from a source PE to a target PE, it must be able to keep track of what PE is and is not responding. This is known as *fault tolerance* and is crucial for any network, given that the probability of PE failure increases as you add more PEs to the network.

C. Remote Direct Memory Access

Remote Direct Memory Access (RDMA) is a form of remote memory access that enables one PE to access, that is to read or write, to memory on another PE without the intervention of the CPU. The alternative to RDMA is remote memory access via *active messages*[8], which are essentially messages that are sent to a handler running on the CPU of the receiving PE, which is used by approaches such as Active Messages over UDP (AMUDP)[9]. The usage of the CPU has two inherent problems: it deprives the PE of valuable computational resources, and it increases overall latency of the operation by the time taken for the CPU to handle the active message. On the other hand, RDMA is handled directly by the NIC, which eliminates not only the issue of resources, but by handling the *direct memory access (DMA)* as soon as the request arrives, it offers a significant reduction in overall latency.

RDMA is commonly used in the *partitioned global address space (PGAS)*, a distributed memory model where the memory of all PEs make up one whole address space, with the exception that the address space is *owned* by, or partitioned to, each individual PE. Addressing schemes typically feature a tuple of virtual addresses and some identifier for the PE, where RDMA requests are sent directly to the virtual address on the target PE. RDMA requests in the PGAS model are referred to as PUT and GET, which are the remote memory analog of a store and load instruction respectively. RDMA is not limited to PUTs and GETs, but

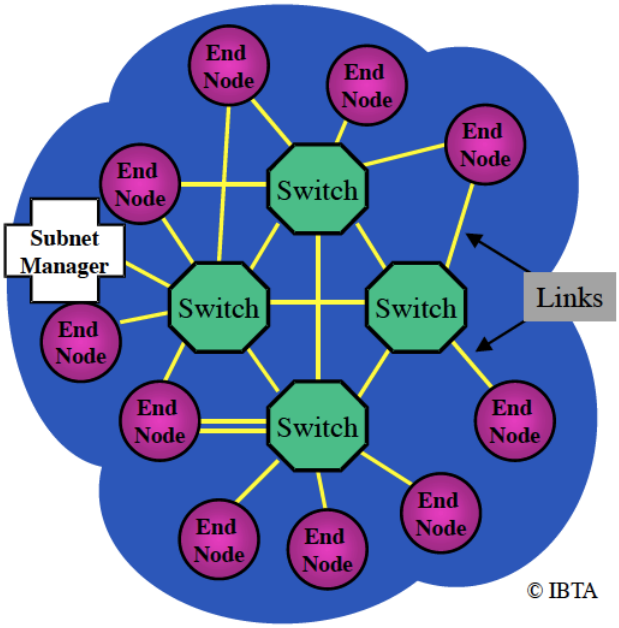

Fig. 4. Infiniband Subnet

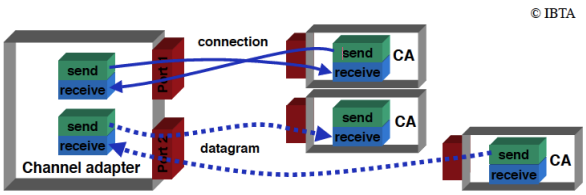

Fig. 5. Infiniband Queue Pair

can allow for RDMA atomic operations as well, where atomics, such as compare-and-swap and fetch-and-add are handled directly by the NIC. The author has empirically utilized RDMA atomics to implement scalable distributed non-blocking data structures, and can personally attest to their usefulness. Finally, you have RDMA to and from devices, and even between devices, such as the case of NVidia's GPUDirect, which enables host-to-remote-device, device-to-remote-host, and device-to-remote-device memory transfers over the network.

III. INFINIBAND

Infiniband, unlike Cray, was originally targeted towards smaller compute clusters and data centers, and is designed to be as cost-effective as possible, while being able to scale up to larger compute-clusters. Where unlike Cray's specialized and custom hardware, Infiniband was designed to work on as many systems as desired. To allow this, it focused on parameterization, allowing multiple *minimum transfer unit (MTU)* sizes, and allowing even major features to be optional, in a strive to ensure that the widest range of configurations are supported. For small clusters, there is "Profile A", while for larger clusters there is "Profile B". Unique to Infiniband is its *verbs*, which is an abstract representation of functionality available on the underlying system. These verbs are unlikely

to be used by most applications, and is instead an abstraction layer used behind-the-scenes by the OS or libraries and APIs.

Infiniband was also created to circumvent the bottleneck in conventional I/O that relied on serial busses, which, while simple, stifle overall bandwidth in a system. Bus-based I/O technology does not scale with the number of devices, nor with the clockspeed of the bus itself, as it pays the penalty of arbitration along the shared bus by each device. As well, busses are primarily used for memory-mapped I/O (MMIO), and generally are interfaced through processor loads and stores to the memory-mapped addressing. While store operation allow reordering around high-latency operations, loads do not, and hence become a severe bottleneck if heavily used, and that on as much as 30% of processor execution time can be spent waiting on I/O loads. Busses also are not very reliable, as a single-device failure can cripple the entire bus, and figuring out which device has failed can be very time-consuming. This bottleneck is non-trivial, as more and more applications are becoming I/O bound, where, for example, data mining techniques require scanning terabytes of at a time, and other applications such as streaming audio and video demand more and more out of the I/O infrastructure. Infiniband is built around replacing bus-based I/O with point-to-point I/O, which provides fault tolerance and scalability, and takes it a step further by modeling all MMIO commands and data transfers between hosts and devices, such as from CPU and GPU, as packets; that is, I/O is treated the same as communication between two hosts or PEs, and by doing so it increases overall bandwidth and reduces latency.

Infiniband's is primarily built around the *subnet*, where each subnet can be joined by a router to create larger Infiniband networks. A subnet is made up of end nodes, switches, links, and a subnet manager, where end nodes, such as PEs and devices, send messages over links to other end nodes, where these messages routed by switches, where the subnet manager defines the routing tables used. A figure of a single Infiniband subnet can be seen in Figure 4[2]. Switches are programmed with forwarding information based on the routing table, where the routing tables may be linear, specifying an output port for each possible destination address, or randomly initialized as pairs of destinations and output ports. Packet sizes are bound by the MTU, and can be 256 bytes, 1KB, 2KB, or 4KBs in size. Infiniband provides several types of communication services between end nodes: *Reliable Connection (RC)* and *Unreliable Connection (UC)* which is similar to TCP/IP, *Reliable Datagram (RD)* and *Unreliable Datagram (UD)* which is similar to UDP, and an Ethernet compatibility mode that uses UDP that allows non-Infiniband transport layers to traverse an Infiniband network. In the case of reliable transport, packets are guaranteed to arrive in order, checked for correctness, with receipt of acknowledgement, barring catastrophic failure. In the case of unreliable transport, each packet contains two separate CRCs, one covering that that cannot change, called the *Constant CRC* and one for data that must be recomputed for data that changes when it leaves the Infiniband network, called the *V-CRC*. As an optimization, all transport is callable and handled from user-mode, meaning no overhead is required to copy data into a buffer to send. RC, UC, and RD classes support RDMA, and even support RDMA on atomics, but is restricted to fetch-and-add and compare-and-swap on 64-bit data.

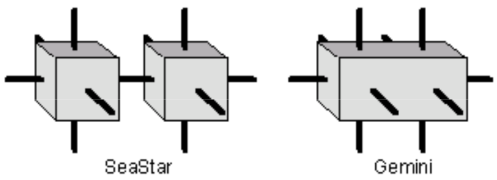

Fig. 6. SeaStar and Gemini ASIC

Channel Adapters, the interface between an end node and a link, communicate using work queues of three types: Send, Receive, and Completion. A Send and Receive Queue are always used as a *Queue Pairs (QP)*, where all messages pass through, as shown in Figure 5. Work generated by some Infiniband verb is placed on a send or receive queue, and when consumed and processed, the completion queue is updated to indicate that the unit of work has finished. Infiniband verbs can send and receive a message and perform RDMA operations which may also specify scatter/gather operations. Memory regions provide a mapping required to perform RDMA using virtual addresses, and memory windows are more fine-grained and lower-overhead constraints on the addressable portion of virtual memory, dictating what can be accessed at the granularity of a byte. Memory windows are registered through Infiniband verbs.

Infiniband provides *Automatic Path Migration* that is able bypass a large class of failures by providing each QP with two independent paths to the destination, where traffic flows naturally across the first path, and only if an error is detected will it switch to the second path. The other end point is also notified of this error and switches re-routes it traffic through the alternative path, and it remains on this alternate path until the original path is repaired. Infiniband also provides *Send Queue Drain*, where a QP will receive packets but will not accept any new requests to send data and instead drains its current send queue of all its packets. When both end point have drained their send queues, then the network is quiescent, meaning no data is traversing across the network, allowing for intervention and repair before the QPs are restarted again. Infiniband clusters routinely use the Fat Tree topology, but recently are using the Dragonfly topology.

IV. Aries

Cray's Aries interconnect is best described as a direct improvement on the Gemini network interconnect. The Gemini *application-specific integrated-circuit (ASIC)* provides not one, but two NICs, and a 48-port router. Each ASIC connects two PEs in a 3D-Torus network, but with a slight variation: instead of having a single switch per PE, such as it was in Cray's SeaStar ASIC[10], it instead connects two PEs together, such as displayed in Figure 6[5], both

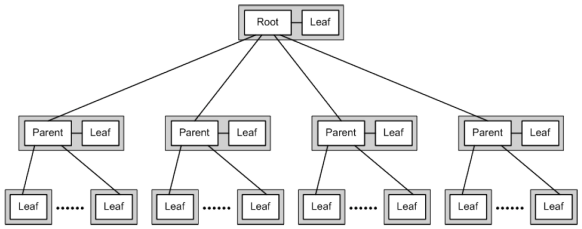

Fig. 7. Aries Collective Engine

eliminating majority of the communication between both PEs and enabling the traffic to and from both PEs to be distributed across the ten links both of the PEs are connected to, not including the internal link connecting both PEs, rather than just six each, allowing for load-balancing on a packet-by-packet basis. For reliability, packet CRCs are checked by each device with automatic link-level retry-on-error, and ECC to protect major memories and data paths within the device. Packet-by-packet adaptive routing is used to distributed traffic over links with a lighter workload, which due to the number of directions that traffic can travel along, it ensures multiple paths are available, optimizing potential bandwidth, and better yet, providing fault tolerance in the event of a PE or link failure.

In Gemini, packets have a triple that makes up the network address, which is composed of the identifier of the PE, a *memory domain handle (MDH)* associated with a memory segment registered at the remote PE, and an offset into the aforementioned memory segment, making up 58 bits, enabling global access to memory anywhere on a PE. The *Fast Memory Access (FMA)* engine handles *network transactions*, which is an abstraction for PUTs, GETs, and *atomic memory operations (AMOs)*, an abstraction for RDMA atomics. FMA translates processor stores into fully qualified network requests and provides low latency. Writes to an FMA descriptor associated with an FMA window determines the remote PE and remote virtual address associated with the base of the window, whereby a write of up to 64-bytes to the put window generates a remote put, and storing a 8-byte control word to the get window generates a get of up to 64-bytes or a fetching AMO, such as a fetch-and-add RDMA atomic. Based on how the FMA descriptor is written to, a scattered write can be performed, and FMA can support both source-side and destination-side synchronization events, which is used to poll the completion of some network transaction. The *Block Transfer Engine (BTE)* handles asynchronous transfers between local and remote memory, but unlike the FMA, which is intended for many small transfers, the BTE is intended for fewer larger ones. A *Completion Queue (CQ)* is used as a lightweight event notification mechanism, which can be polled on to be notified of the completion of a FMA or BTE network request. AMOs supports a much wider and richer range of RDMA atomics compared to Infiniband, as shown in Table I, which provides an almost one-to-one translation of processor atomics to RDMA atomics. Indeed, an AMO cache is even maintained, which reduces the needs for reads of local memory when multiple processes, remote or local, access the same atomic variable, where writes to local memory, including from incoming RDMA atomics, update the actual cache first and then actual memory *lazily*. The AMO cache provides a great speedup for RDMA operations, but come with a significant penalty in that all accesses to memory updated by RDMA atomics *must* go through RDMA atomics due to the lack of coherency, which comes with significant overhead, in which the author measured to be about one to two orders of magnitude in certain applications.

Gemini provides a 16-bit packet CRC, which protects up to 64-bytes of data and the associated header, and major memories are protected using ECC. Gemini links use a sliding window protocol, where the receiving the link checks the CRC as a packet arrives, where an error is returned if it is incorrect, causing the sending link to retransmit on receipt of the error. The CRC is also checked as a packet leaves each PE, and again as it transitions from the router to the NIC. Completion events include details of the status of each transaction, which allows the software to recover from errors. In the event of link failure, adaptive routing will maneuver around it, and in the event of the loss of connectivity between two PEs, the network quiesces, halting all flow through the network, computes new routing tables, and then re-enables the network.

Aries, unlike Gemini, uses four NICs, where a single switch handles connectivity to four PEs, which is exactly double what Gemini provided. Where Gemini was used in the Cray-XT series, Aries is used in the newer Cray-XC series, and uses the Dragonfly topology rather than the 3D-Torus that the Gemini had. Often, the *bijection bandwidth*, which is measured as the worst-case bandwidth achieved by dividing a system in two, with all PEs in each half communicating with a peer in the other half, is compared to the injection bandwidth to measure the ability of an HPC network. In the 3D-Torus topology used in Cray XE6 systems, there is five times as much routing bandwidth as injection bandwidth, but performance degrades when packets take an average of more than five hops, and the all-to-all bandwidth falls as the system size increases. Aries also adds a new *Collective Engine (CE)*, which provides hardware support for reduction and barrier operations, and is optimized for latency sensitive, single-word operations such as 8-byte integers. The CE supports reduction trees as shown in Figure 7[3] that ensure scalability which is created either during job initialization or potentially later on-demand. Each PE joins a reduction operation, which offloads the work to the NIC, supplying its contribution and passing partial results up the tree towards the root, which then the final result is scattered back down the tree and written to each of the participating PEs, generating a completion event for each PE. Unfortunately, Aries has not brought any support for NVidia's GPUDirect, causing it to fall out of favor for GPU-heavy workloads such as deep learning.

V. Performance

The performance of both Infiniband and Aries can only truly be explained by analyzing actual data, and so the

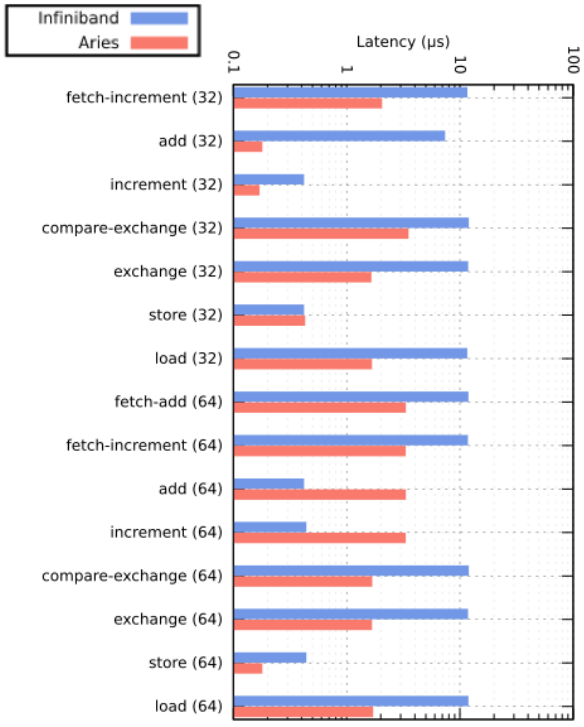

Fig. 8. Latency of RDMA Atomics

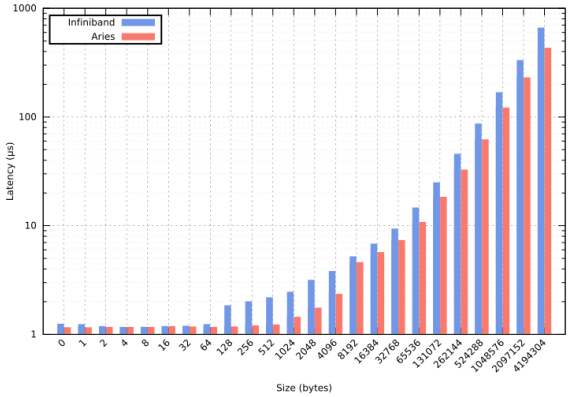

Fig. 9. Point-to-Point Latency

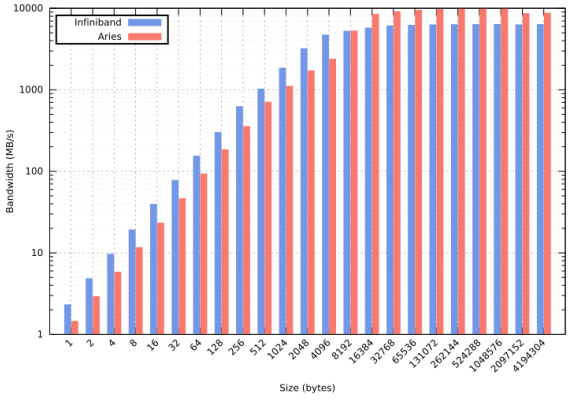

Fig. 10. Point-to-Point Unidirectional Bandwidth

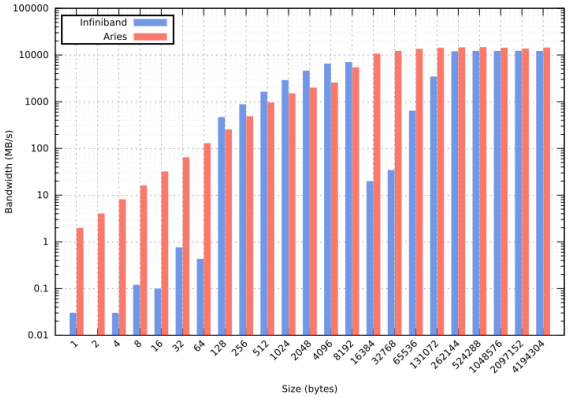

Fig. 11. Point-to-Point Bidirectional Bandwidth

results of running microbenchmarks provided by Ohio State University (OSU), which tests point-to-point, all-to-all collective, and one-sided communication in a network. The results for Aries was obtained on a Cray-XC50 supercomputer with Intel Broadwell 44-core PEs using CrayMPI and CraySHMEM, and the results for Infiniband was obtained on an Infiniband cluster with Intel Xeon 20-core PEs using OpenMPI and Sandia-OpenSHMEM, both networks capping out at 10GB/s of bandwidth. Each test was run with only two PE, the minimum required, selected by Torque and Slurm respectively, which may not entirely account for the distance between each node. Of the microbenchmarks available, only the point-to-point latency (Figure 9), unidirectional (Figure 10) and bidirectional bandwidth (Figure 11), and OpenSHMEM RDMA atomics (Figure 8) are provided.

Point-to-Point latency is tested by pairing both PEs and having them initiate blocking MPI_Send and MPI_Recv in a ping-pong fashion. In terms of point-to-point latency, for smaller messages up to 64 bytes, both Infiniband and Aries are about equal, until it exceeds the maximum size for a packet's payload, where the latency jumps as much as 1.5x. Infiniband, from this point, consistently has 1.5x more latency than Aries for the remainder of the benchmark. Point-to-Point bandwidth is tested where, in the unidirectional benchmark, one PE sends enough data to fill the receiver's window, and then waits a reply from the receiver, where in the bidirectional benchmark, both PEs both send data to fill the other PEs window, and sends a reply, using the non-blocking MPI_ISend and MPI_IRecv for both microbenchmarks. For unidirectional bandwidth, Infiniband has the advantage for small to medium sized messages, but likely due to Aries BTE being built for asynchronous large transfers rather than synchronous, this may be an optimization of Infiniband, but Aries does seem to come out on top at the end, hitting the ceiling on its bandwidth while Infiniband hits a bottleneck that isn't the maximum transfer rate itself. In the bidirectional case, Aries is a clear winner in terms of both stability and speed, as can be observed by the identical performance Aries gets, while Infiniband drops by

almost two orders of magnitude.[1] RDMA atomics is tested by paring both PEs and having one PE initiate OpenSHMEM RDMA atomic operations on another and collect the average latency. While Infiniband only supports Fetch-And-Add and Compare-And-Swap, the implementation of OpenSHMEM is free to implement the rest of the operations with either of the two avaiable RDMA atomics or as active messages. The only RDMA atomic operation that Infiniband outperforms Aries is on the 64-bit add and increment, although it is not clear as to why.

VI. Future of Networks in High-Performance Computing

In an interview[11] with Steve Scott[2], Chief Technology Officer (CTO) of Cray Inc., a Hewlett Packard Enterprise company, has led to some information and details surfacing on Cray's Shasta machines, the Slingshot interconnect[12], and Rosetta ASIC. Steve Scott, who also happened to be instrumental in the development of the SeaStar interconnect, which came before Gemini, is taking another hands-on approach to the development of the brand new interconnect by Cray. Not only will the interconnect come with native support for Ethernet so that it can be used in data centers, but it provides its own state-of-the-art protocol suitable for HPC. Slingshot is to provide adaptive routing, congestion control, and other features necessary for HPC networks. Its design has been motivated by increasingly heterogeneous workloads that are common in data centers, simulations, data analytics, and machine learning, and is built to handle all of these simultaneously.

Shasta is to handle a broad range of processor types, node sizes, and even be power-efficient with its own direct liquid cooling, while also allowing the Slingshot interconnect to be used in the standard 19-inch racks where air-cooling is used. Gone are the days of it being all-or-nothing, where you had to own a Cray-XC to use Aries or Cray-XT to use Gemini, and welcome are the days of flexibility. Shasta is designed to allow the connection of storage devices to the Slingshot interconnect, similar to how Infiniband does, but in a way that saves cost and provides a lot better small I/O performance. The highly configurable interconnect can even vary the injection bandwidth into nodes and the amount of global bandwidth in the system, and is designed to scale to "exascale and beyond". Slingshot is an Ethernet ASIC, with all of the qualities of an HPC network: "smaller packets, smaller packet headers, reliable hardware delivery across the link, credit-based flow control..." The switch ASIC is called Rosetta, which implements 64 ports, with an individual throughput of 200 GB/s, and an aggregate throughput of 12.8TB/sec. Slingshot uses a Dragonfly topology, but has only a single dimension within each group, resulting in at most three hops between any two endpoints in the network, at 300 nanoseconds per hop, with 100 nanoseconds being used for forward error correction. This high-radix and low-diameter network is truly designed as an exascale interconnect.

Congestion control has been built around dramatically reducing queueing latency in the network, and providing *performance isolation* between workloads, which it does so by bookkeeping what is transferred between every set of endpoints in the network, allowing it to quickly detect congestion and *backpressure* the offending source in a way that does not impede the traffic for the victim. In what Steve Scott refers to as *tail latency*, which is the latency of the slowest packet, these tail latencies can be flattened to become more uniform with the implemented congestion control. Tail latencies, as Steve Scott has stated, can get exceptionally bad in data centers where they drop packets out of fear of (fat-) tree saturation, and also in applications with a load imbalance with regard to network traffic, and so Slingshot can be very useful for aforementioned applications.

[1]This is likely a quirk of the individual network itself, and may not be present in more recent and advanced Infiniband compute clusters.

[2]Steve Scott happens to be the brother of the author's advisor, Michael L. Scott.

VII. Conclusion

The properties of a high-performance network have been summarized, and it comes down to their ability to network topology, which by itself controls latency and bandwidth, the fault tolerance and reliability, which is crucial for performance as much as correctness, and then communication primitives such as RDMA. Three different network topologies, the Fat Tree, Torus, and Dragonfly have been surmised in brief, as well as some information and commonly used methods for providing reliability and details as to why topology determines the fault tolerance of a network. RDMA, in terms of the PGAS primitives of PUT and GET have been defined, as well as their atomic counterparts, and finally a note on RDMA concerning GPUs were given. The design and implementation of Infiniband was discussed, as well as Gemini, predecessor Aries, and then Aries itself was looked at in detail. Performance obtained on the OSU benchmarks from a Cray-XC50 supercomputer and Infiniband compute cluster were gathered and analyzed, and then details on some recent insight from the CTO of Cray Inc., a Hewlett Packard Enterprise company, Steve Scott, was examined.

References

[1] G. F. Pfister, "An introduction to the infiniband architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, pp. 617–632, 2001.

[2] "Introduction to InfiniBand™ — Mellanox Technologies." [Online]. Available: https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf

[3] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," *Cray Inc., White Paper WP-Aries01-1112*, 2012.

[4] R. D. Chamberlain, M. A. Franklin, and Ch'ng Shi Baw, "Gemini: an optical interconnection network for parallel processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 10, pp. 1038–1055, Oct. 2002.

[5] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini System Interconnect," in *2010 18th IEEE Symposium on High Performance Interconnects*, Aug. 2010, pp. 83–87.

[6] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over Commodity Ethernet at Scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 202–215, event-place: Florianopolis, Brazil. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934908

[7] "Github repository: Hewlettpackard/genz_tools_network_monitoring." [Online]. Available: https://github.com/HewlettPackard/genz_tools_network_monitoring

[8] T. Eicken, D. Culler, S. Goldstein, and K. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," in *[1992] Proceedings the 19th Annual International Symposium on Computer Architecture*, May 1992, pp. 256–266, iSSN: null.

[9] D. Bonachea and D. Hettena, "AMUDP: Active Messages Over UDP," p. 21.

[10] R. Brightwell, K. Pedretti, K. Underwood, and T. Hudson, "SeaStar Interconnect: Balanced Bandwidth for Scalable Performance," *IEEE Micro*, vol. 26, no. 3, pp. 41–57, May 2006.

[11] T. P. Morgan, "Cray Slingshots Back Into HPC Interconnects With Shasta Systems," Oct. 2018. [Online]. Available: https://www.nextplatform.com/2018/10/30/cray-slingshots-back-into-hpc-interconnects-with-shasta-systems/

[12] "Slingshot: The Interconnect for the Exascale Era," p. 6.