An Evaluation of the CORAL Interconnects

Christopher Zimmer[†], Scott Atchley[†], Ramesh Pankajakshan[*], Brian E. Smith[†], Ian Karlin[*], Matthew L. Leininger[*], Adam Bertsch[*], Brian S. Ryujin[*], Jason Burmark[*], André Walker-Loud[◇], M. A. Clark[§], Olga Pearce[*]
{zimmercj,atchleyes,smithbe}@ornl.gov
{karlin1,leininger4,pankajakshan1,bertsch2,ryujin1,burmark1,pearce8}@llnl.gov
walkloud@lbl.gov,mclark@nvidia.com
[†]Oak Ridge National Laboratory,[*]Lawrence Livermore National Laboratory, [◇]Lawrence Berkeley National Laboratory, [§]NVIDIA Corporation

ABSTRACT

The US Department of Energy deployed the Summit and Sierra supercomputers with the latest state-of-the-art network interconnect technology in 2018 and both systems entered production in 2019. In this paper, we provide an in-depth assessment of the systems' network interconnects that are based on Enhanced Data Rate (EDR) 100 Gb/s Mellanox InfiniBand. Both systems use second-generation EDR Host Channel Adapters (HCAs) and switches with several new features such as Adaptive Routing (AR), switch-based collectives, and HCA-based tag matching. Although based on the same components, Summit's network is "non-blocking" (i.e., a fully provisioned Clos network) and Sierra's network has a 2:1 taper between the racks and aggregation switches. We evaluate the two systems' interconnects using traditional communication benchmarks as well as production applications. We find that the new Adaptive Routing dramatically improves performance but the other new features still need improvement.

CCS CONCEPTS

• **Networks → Network performance evaluation**; **Network performance analysis**.

KEYWORDS

High Performance Computing, interconnect, InfiniBand, EDR, latency, bandwidth, switch collectives, congestion, tag matching, offload

ACM Reference Format:
Christopher Zimmer[†], Scott Atchley[†], Ramesh Pankajakshan[*], Brian E. Smith[†], Ian Karlin[*], Matthew L. Leininger[*], Adam Bertsch[*], Brian S. Ryujin[*], Jason Burmark[*], André Walker-Loud[◇], and M. A. Clark[§], Olga Pearce[*]. 2019. An Evaluation of the CORAL Interconnects. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3295500.3356166

1 INTRODUCTION

In 2012, the US Department of Energy (DOE) formed the Collaboration of Oak Ridge, Argonne, and Livermore (CORAL), a joint procurement effort to acquire three leadership class supercomputers. Oak Ridge and Livermore deployed and accepted the first systems, Summit and Sierra, in 2018. The systems are based on the IBM AC922 [17] node design. The nodes contain IBM POWER9 processors, NVIDIA V100 GPUs, a 1.6 TB Samsung NVMe storage device, and two ports of EDR InfiniBand. Both systems entered into production in early 2019. Summit and Sierra have held the number one and two positions on the Top500 [12] since November 2018.

The Oak Ridge Leadership Computing Facility's, (OLCF) Summit system contains 4,608 compute nodes. The system fabric is a single plane, "non-blocking" fat-tree[1] where both compute and storage are integrated into the same fabric. The network design, with full global bandwidth, caters to open-science, "capability" applications exploring the natural world. OLCF defines a capability application as one that uses between 20% and 100% of Summit's nodes.

Livermore Computing's (LC) Sierra system contains 4,320 compute nodes. The system has a 2:1 tapered fat-tree network. To create the taper, each edge switch has one connection to an aggregation switch for every two connections to compute nodes. LC chose this approach based on the capacity (less than 20% of system) and capability job size data as well an understanding of the messaging patterns of key applications that are expected to use Sierra. The money saved on the network allowed LC to procure additional nodes that are expected to increase overall throughput more than the performance lost through the network tapering.

In this paper, we seek to answer several questions regarding the interconnects of these two large systems. First, how does the addition of new InfiniBand technologies impact production applications and benchmarks? Mellanox EDR InfiniBand has introduced several significant features to accelerate various aspects of an HPC application. These new features include:

- Adaptive Routing (AR): the ability to re-route packets mid-traversal to avoid congestion,
- Scalable Hierarchical Aggregation and Reduction Protocol (SHARP): an HPC collective optimization moving collective processing into the switch and,

[1]While Summit's network is a Clos network, we use the more common term "fat-tree" to describe the topology of which Clos is a subset. Also, "non-blocking" is a theoretical statement about a Clos network and Clos networks do experience congestion as we will show.

- Hardware Tag Matching: an offloading of the MPI tag matching algorithms to the HCA's ASIC.

The second question seeks to address the impacts of system topology differences. The Sierra and Summit systems, while employing the same technologies, have slightly different configurations due to their respective workloads. Sierra uses a 2:1 tapered fat-tree while Summit uses a 1:1 "non-blocking" fat-tree. From a high-level perspective, the tapered network has reduced global bandwidth and fewer paths for adaptive routing. The impact of this design choice on applications and benchmarks is not well understood with this generation of network technology. We measure the impacts and provide initial user experiences.

Finally, as these are production systems, we seek to understand how these large networks perform under congestion. We study tail-latency and its potential magnitude given the changes in technology and the choices of topology. Tail-latency has the potential to significantly disrupt bulk-synchronous applications. These two systems have an unexpected view of the network ports and the MPI library provides many choices for using these ports. We evaluate many of these choices and show their worst-case impacts.

The rest of the paper is organized as follows: Section 2 discusses the new features in Mellanox's EDR ConnectX-5 and SwitchIB-2. Section 3 provides topology comparisons between current systems at LLNL and ORNL; Section 4 discusses the performance of traditional networking micro-benchmarks using the new features, a new tail latency benchmark from Cray that measures the impact of congestion, as well as production DOE applications using some of the new features; Section 5 shares some early production application experiences, and finally Section 6 discusses our findings and recommendations for users and system architects.

2 INTERCONNECT DESIGN AND FEATURES

The design of the compute fabric for the CORAL systems focused on a few particularly challenging acceleration tasks. In this section, we provide more detail on the communication accelerations, adaptive routing, Scalable Hierarchical Aggregation and Reduction Protocol, hardware tag matching and the design choices behind them.

2.1 Spectrum MPI

Both systems use IBM's Spectrum MPI [18] as the primary communications software. Spectrum MPI is based on an OpenMPI 2.x base implementation. Process management in Spectrum MPI is handled through PMIx from OpenMPI. The low-level messaging interface is implemented through Parallel Active Message Interface (PAMI) [32].

2.2 Adaptive Routing

Historically, Mellanox InfiniBand networks have been statically routed except in the case of failure. In the case of path failure, either due to link or port failure, the network re-routes. This traditionally is expensive and requires several steps. The subnet manager must detect the failure, quiesce the network, calculate new network paths to omit the failed ports, flush the caches, and distribute the new paths before reactivating the network. At least one reason for this expense was that InfiniBand networks were strictly in-order networks. Re-routing flows dynamically would violate the constraints of the network leading to data-corruption. With the release of EDR, in particular, ConnectX-5 and Switch-IB 2, Mellanox's InfiniBand networks now support out-of-order packet receipt. To maintain traditional InfiniBand semantics, Mellanox's software ensures that completions are released to the application in-order.

Adaptive Routing [25] enables the re-ordering of InfiniBand packets mid-flight to bypass congestion. The basic implementation of AR builds upon the notion commonly used in switches called a Linear Forwarding Table (LFT). The LFT is the mapping of a terminal end-point to an egress port on a switch. When a packet arrives, a quick lookup on the LFT tells the switch which egress port to enqueue the packet for its next transmission. AR extends this concept for topologies that offer multiple paths between two end-points in the network. This is true of both non-blocking and 2:1 tapered fat-trees. AR essentially adds a supplemental AR-LFT that maps several egress ports to a destination group. Packets are then distributed across these egress queues based on queue depth as a local measure for congestion.

2.3 Scalable Hierarchical Aggregation and Reduction Protocol

The Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) [15] was introduced as a feature in SwitchIB-2 and is used to accelerate barrier, broadcast, reduce, and allreduce collective operations. This feature is intended to reduce the cost of large-scale collective operations that traditionally dominate large job communication overheads. SHARP essentially works by allocating an overlay tree across switch resources used in the network. Each switch is provisioned with memory and processing capability. The overlay tree performs the collective operation at the switch level, including any computation and broadcasts that are necessary. The acceleration associated with SHARP is the result of almost a 50% reduction in communication as nodes only communicate up the overlay tree with switches as the destination.

There are several considerations when using SHARP on large systems like Sierra and Summit. SHARP is a shared resource that is not currently integrated with any scheduler resources. Thus, jobs are unable to pre-provision or specify the specific amounts of fabric resources that they will need. SHARP does offer the ability to statically allocate resources, called outstanding transactions (OST)s based on the job-size. However, this is a static policy and will lead to allocations of resources for jobs choosing not to use SHARP. Another consideration is that SHARP resources are associated with communicator groups, each additional communicator group potentially increases the number of OSTs needed to satisfy the application's collective acceleration.

SHARP operates as an on-node daemon (sharpd) and the SHARP Aggregation Manager (sharp_am). At `MPI_Init()`, sharpd will calculate the initial set of OSTs available to the job through sharpd and sharp_am. Due to the variable availability of the amount of resources a job can receive, SHARP has a fall-back to software collectives as well as a mode that will fail the application when resources are requested but cannot be secured. This could pose challenges for applications that experience long queue times to contend with slower collective performance or have the job not run.

This can be mitigated using static allocation, but this hardlimits the number of OSTs that any job can use.

2.4 Hardware Tag-Matching

High performance computing depends on the Message Passing Interface (MPI)[14] for inter-process communication. One of the key elements of MPI is tag-matching for two-sided (i.e., send and receive) communication. MPI defines an "envelope" as a tuple that specifies the *communicator*, the *sender's rank*, the *receiver's rank*, and an integer *tag*. A process' rank is its index within the communicator.

Most MPI's perform tag matching in software when either the application calls into the MPI library or via a background thread. Applications may call `MPI_Test()` to poll for the completion or `MPI_Wait()` to block for the completion. Regardless of the method (polling, waiting, or background thread), modern MPI implementations will internally spin, calling the underlying network's completion queue to minimize latency. The implementation could choose to internally wait (sleep) using a notification to wake it when a message arrives, but this can add 1-2 μs to the latency. Several additional methods have been proposed to improve progression including running progress threads on dedicated cores [20], collaborative polling [10], cooperative rendezvous [5], and optimized async progress [30], but none are employed by Spectrum MPI. By default, Spectrum MPI only processes tag matching when the application calls MPI_Test, MPI_Wait, or variants. It has an option for a background progress thread that will perform matching, but it can consume a core per process.

The Mellanox ConnectX-5 HCAs integrates tag-matching into a hardware offload. The goals of hardware Tag-Matching (TM) are to (1) reduce latency for two-sided messaging (or at least be no worse), (2) provide automatic progression of rendezvous sends, and (3) to reduce CPU power consumed while performing matching (i.e, to avoid a CPU thread spinning on the Verbs completion queue).

3 OLCF AND LC SYSTEM TOPOLOGY COMPARISONS

Both systems employ a three-level fat-tree described later in this section. The largest difference between the Sierra and Summit networks is that Summit has a "non-blocking" fat-tree network while Sierra has a 2:1 tapered network. The design choice differences were driven by different workloads, job size and compute node design at each site.

As an open science machine, Summit must be designed to handle a diverse set of use cases that can vary year to year based on a grant based allocation policy. Many of these applications use methods such as QCD and 3D FFTs which require significant network bandwidth. Sierra serves a critical mission role for the National Nuclear Security Agency (NNSA) and has been designed specifically to handle NNSA codes and workloads. Recent studies at LLNL have looked into typical messaging patterns and concluded that a 2:1 tapered network suffices for capacity computing workloads on CPU-based architectures[21]. This study was reanalyzed, in the context of the expected Sierra workloads and architecture, and 2:1 network tapering was determined to be reasonable for the Sierra system. Within a fixed platform budget, the Sierra 2:1 network tapering enabled the acquisition of additional compute nodes that enhanced overall workload throughput.

While Summit is designed to handle jobs that take 20-100% of the machine, Sierra is expected to run a combination of capacity and capability workloads. LLNL observed that the Sequoia system workload consisted of jobs that utilize 25% of the system or less consuming 70% of the cycles, and larger jobs consuming the remaining 30%. Overall the typical job on Sierra is expected to be smaller than on Summit. In addition, historical data shows that jobs on LC systems tend to use about the same percentage of the next machine as the previous one.

Sierra and Summit utilize the same POWER9 CPU and V100 GPU processors, however, Summit has six GPUs per node while Sierra has four [35]. With more compute per node, codes effectively using Summit's GPUs will have higher communication needs than Sierra.

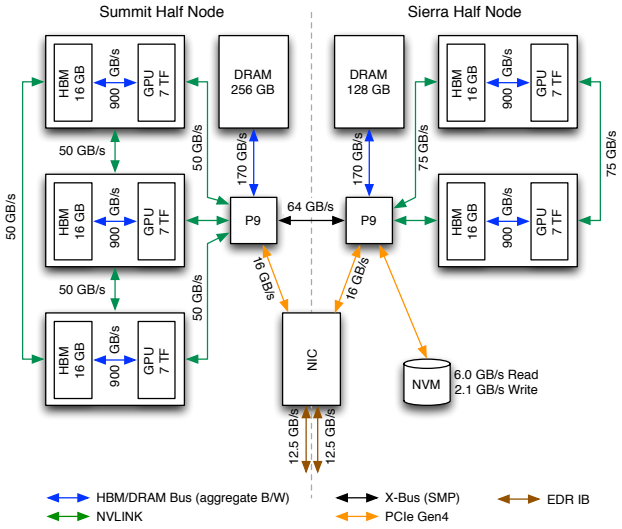

Figure 1: Node block diagram. One half of a Summit is shown on the left and one half of a Sierra node is shown on the right. Both nodes have the NVMe device. The shared HCA is shown in the lower center.

Both Summit and Sierra use the same basic interconnect components: HCAs, rack switches, and aggregation switches. Each node has one HCA in a PCIe Gen4 x16 slot. This slot is electrically connected to both POWER9 CPUs such that each host has eight lanes of PCIe capable of 16 GB/s. Figure 1 shows the block diagram for both nodes. The HCA (labeled NIC) connects directly to both CPUs and both CPUs can directly use both ports. When Linux enumerates the PCI tree, each CPU reports both ports on the HCA so Linux recognizes four ports. Socket 0 maps virtual ports V0 and V1 to the physical ports P0 and P1 and socket 1 maps virtual ports V2 and V3 also to P0 and P1 as shown in Figure 2. The figure also shows socket 0 striping over virtual ports V0 and V3. In this case, the V3 data will cross the SMP bus from socket 0 to socket 1 and then down PCIe to physical port P1. By default, processes on Summit do not

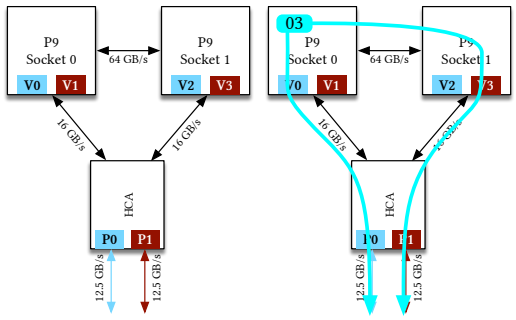

Figure 2: The left image shows the mapping of virtual to physical ports for both sockets. The right image shows socket 0 striping data over virtual ports V0 and V3.

stripe and all socket 0 traffic uses virtual port V0 and all socket 1 traffic uses virtual port V3.

The rack switches (commonly referred to as Top Of Rack or TOR switches) have 36 ports. In both systems, there are 18 "down" connections per switch to compute node HCAs. In Summit, there are 18 "up" links, one to each of the 18 aggregation switches. In Sierra, there are nine "up" links to the nine aggregation switches and nine empty ports. Within the same rack switch, both systems are "non-blocking" when communicating between ports connected to the same rack switch. The taper only impacts communication that goes up to an aggregation switch. On both systems, each rack has two rack switches. One rack switch connects to physical port P0 and the other rack switch connects to physical port P1.

An aggregation switch, or Mellanox Director class switch, is a two-level, "non-blocking" Clos network[7] in a single chassis with the rack switches comprising the third-level of the network. These switches have 18 "spine" switches, sitting at the top of the tree, which connect internally up to 36 "leaf" switches. As described above, rack switches connect to leaf switches of the aggregation switches. A fully configured SwitchIB-2 (EDR) Director switch has 648 ports. In both systems, groups of 18 racks have a single connection to the same set of 18 (Sierra) or 36 (Summit) leaf switches. Using the default Up*/Down* routing[33], traffic within this subtree does not need to go up to the spine switches. Each leaf-level subtree is a two-level fat-tree (leaf switch and rack switches) requiring three switch hops.

4 EVALUATION

In 2018, the integration teams fielding Summit and Sierra published traditional micro-benchmarks for the system including the network interconnect[35]. The MPIGraph [27] results showed benefits of Adaptive Routing in isolation, and the SHARP tests showed nearly flat scaling for an eight byte `MPI_Allreduce()` up to 2,048 nodes. This SHARP performance was 74% faster than IBM's software collectives. SHARP also performed well on `MPI_Barrier()` with ~8μs for SHARP compared to ~35μs for Spectrum MPI. In this section, we evaluate the impact of these new features using additional benchmarks as well as production applications. We retest some of the benchmarks at larger scales and evaluate features that were not covered in the previous paper.

4.1 Adaptive Routing

4.1.1 Micro-benchmarks. Mellanox promotes Adaptive Routing as a mechanism for improving the bandwidth available to applications by increasing path diversity between two end-points in the network. We quantify the impact on bandwidth using two common HPC benchmarks, MPIGraph and the ALCF [37] bisection measurement test. The first application, MPIGraph, uses a single rank per node in an advancing ring communication pattern. At each iteration, each rank increases their communication partner by one, until all ranks have communicated with all other ranks. At the full scale of Summit, this resulted in over 21,000,000 measurements between the ranks. The results of this measurement are shown in Figure 3 as a histogram of every measurement taken. The MPIGraph benchmark was run under two routing conditions - once using traditional IB static routing and again with adaptive routing. Our results show both larger and more consistent bandwidth measurements with adaptive routing. Adaptive routing achieves over double the average bandwidth of the statically routed measurements. Performance clusters between 16 and 19 GB/s while the statically routed measurements range from 4 to 12 GB/s.

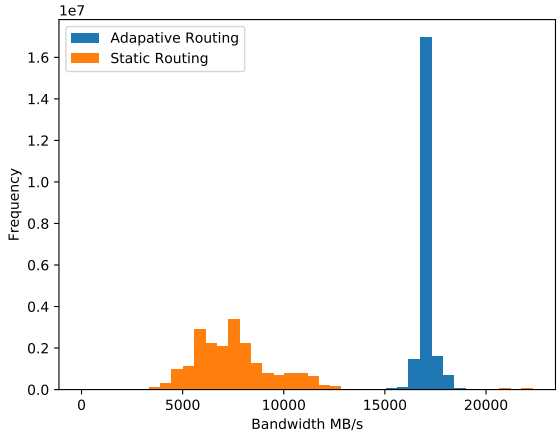

Figure 3: Summit: MPIGraph histogram of all send measurements adaptively routed and statically routed.

Figure 4 shows the results for the same test run on Sierra. As with Summit, Sierra benefits with AR performance clustering around 10 GB/s while the statically routed results have a wider distribution near 5 GB/s. Sierra's AR data is not as tightly clustered as Summit's. This cluster is spread over two bins, each of which has about half the frequency as the single bin on Summit.

We ran a second test using the ALCF bisection measurement tool on 4,608 nodes of Summit with and without adaptive routing. The benchmark was configured to use six MPI ranks and to stripe large data messages across both HCA ports. The test utilized a striping policy on the node with data crossing the SMP bus, thus avoiding PCI bottlenecks. For the static routing case, the bisection bandwidth was 39,337 GB/s or only 34% of peak. While the adaptive routing case, the bisection bandwidth was nearly 2.4x higher at 95,592 GB/s

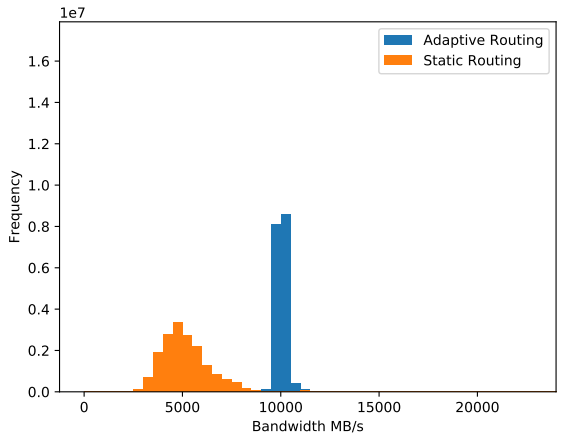

Figure 4: Sierra: MPIGraph histogram of all send measurements adaptively routed and statically routed.

and nearly 83% of peak bandwidth. Additional impacts of Adaptive Routing are evaluated in the congestion study in this section.

4.2 SHARP

4.2.1 Micro-benchmarks. The first set of SHARP measurements were taken during the Summit acceptance phase to evaluate collective performance relative to Summit statement-of-work (SOW) technical requirements. The results are shown in Table 1 and compare Spectrum MPI configured with either the IBM collective library, the Mellanox HCOLL software library, or the Mellanox SHARP hardware collectives. The results demonstrate a 2x or greater reduction in latency when using the SHARP collectives.

Collective	Nodes/PPN	SMPI (μs)	HCOLL (μs)	SHARP (μs)
Barrier	4096/1	20.40	50.19	9.50
8b Allreduce	4096/1	23.91	23.30	7.79
2k Allreduce	4096/1	55.57	57.65	21.85

Table 1: OSU Collective Benchmark: Latency is significantly reduced when using SHARP on 4096 nodes.

We also evaluated a Conjugate Gradient solver benchmark introduced in [36]. This benchmark consists of repeating `MPI_Allreduce()` calls with compute iterations between Allreduces. The benchmark was configured to run on 512 nodes using 42 processes per node and for 512,000 iterations, for a total of 1,024,000 allreduces. Each allreduce was a single 8-byte (double precision) value.

Using tuned command-line parameters, provided by Mellanox [26], the overall benchmark walltime decreased from 18.80 seconds to 12.66 seconds, an improvement of almost 33%. Nearly half the benchmark walltime is spent in the `MPI_Allreduce()` calls. We investigated adding an `MPI_Barrier()` before each allreduce to ensure all processes were synchronized before entering the allreduce. However, this decreased performance considerably - walltime was 32.8 seconds for the non-SHARP case and 20.67 seconds for the SHARP case.

4.2.2 Miniapps. SHARP shows significant benefit to micro-benchmarks where the runtime is dominated by a particular collective operation. Now we evaluate the impact in mini-applications with mixed messaging workloads.

AMG is a parallel algebraic multigrid solver for linear systems for problems arising from unstructured grids. It is an extraction of the BoomerAMG solver [16] from the Hypre library with a driver added to allow it to function as a standalone code. Communication is dominated by small point to point messages and frequent eight-byte AllReduces [22].

We ran AMG ten times on both Sierra and Summit using one rank per GPU, four and six GPUs respectively. Figure 5 shows performance for 128 nodes on both systems. In each run, turning on SHARP reduces performance noticeably. In Figure 6, we ran AMG ten more times on Summit using 256 nodes hoping that strong scaling the problem would increase the communication time relative to the compute time. Again, we see reduced performance when SHARP is enabled.

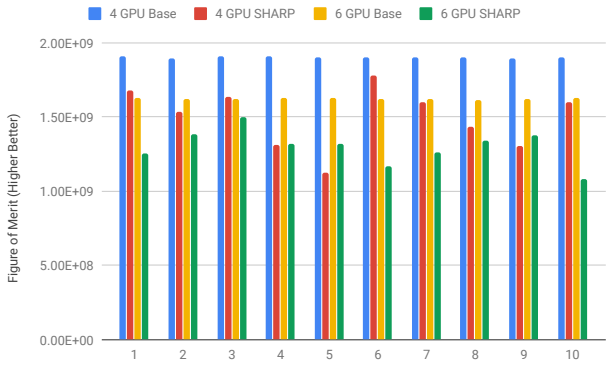

Figure 5: AMG 128 nodes measurement comparing SHARP performance to Spectrum Software Collectives.

Nekbone is a mini-app that represents the computationally intensive conjugate gradient solver in the Nek5000 application [13]. Similar to AMG, the communication is dominated by nearest neighbor exchanges and AllReduces. Unlike AMG, Nekbone is a balanced calculation so the AllReduces should be more synchronous. Typical runs show 1-2% of the overall walltime are spent in communications in contrast to the CG benchmark where almost 50% of run-time is spent in communications. Nekbone has a single 8 byte MPI_Allreduce which is called a number of times and multiple point-to-point messages of various sizes. Nekbone is set to show weak scaling. As process counts increase, more total work is done so run times are reasonably constant with process count. In this case, the problem size was set to have roughly two minutes of run time. We ran from 32 to 512 nodes. All tests were with six MPI processes per node, each process using one GPU. All tests were run six times and the walltime was averaged. SHARP showed a 1-2% decrease in performance at smaller node counts (32 to 256 nodes) and a 1-2% increase in performance at 512 nodes. However,

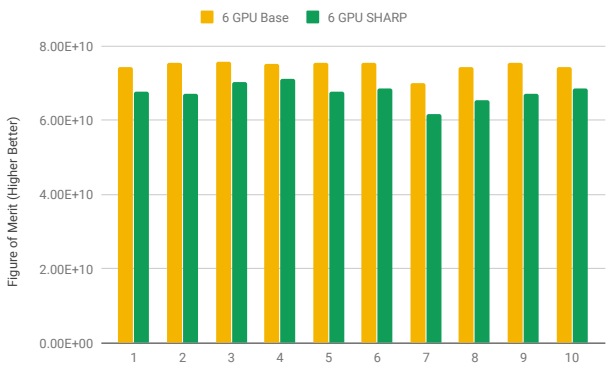

Figure 6: AMG 256 node performance on Summit comparing SHARP performance to Spectrum Software Collectives.

Nekbone shows roughly 1-2% variation run-to-run so it is not clear that SHARP had an impact (positive or negative) on results.

4.3 Hardware Tag-Matching

4.3.1 Micro-benchmarks. Two separate benchmarks were used to study the impact of hardware MPI tag-matching. The hardware tag-matching feature offloads MPI tag-matching, thus shifting the operation from the POWER9 processor to the Mellanox HCA. This impacts small eager messages and larger message rendezvous progression. We evaluate the impact with the osu_latency benchmark run between two nodes with and without hardware tag-matching enabled.

Enabling tag-matching in Spectrum MPI on Summit and Sierra is handled through two flags and an environment variable:

```
--smpiargs="-mca pml_pami_use_2sided 1"
-E PAMI_IBV_ENABLE_TAG_MATCHING=1
```

The first flag switches PAMI from using the active messaging API to using the two-sided API. The second flag enables PAMI's software emulation layer to perform tag-matching. The environment variable, PAMI_IBV_ENABLE_TAG_MATCHING, shifts tag-matching from the software layer to the hardware. The combination of these flags and environment variable enables hardware tag-matching.

The results shown in Table 2 show the latency impact when using hardware tag-matching for eager messages up to 8 KiB. For smaller messages (<256 bytes), there is a slight increase in latency (~10-60 ns). For messages between 256-8,192 bytes, hardware tag matching lowers the latency from 10 ns at 256 bytes to 310 ns for 8 KiB.

We measure the second feature of hardware tag-matching, rendezvous offloading, using MPI_Overhead [11] from the Sandia MPI Micro-Benchmark Suite. The MPI_Overhead benchmark determines overhead using a post-work-wait loop. As the primary loop iterates, a work time (work_t) is gradually increased during each iteration prior to calling into MPI_Wait. When this value is small the overall loop time does not increase. However, after a point, the work time will exceed the MPI_Wait time. At this point, the increased loop time minus the work time is calculated as the overhead time.

Message Size	Base Latency	HW Tag Matching	Higher/(Lower) Latency (ns)
0	1.20	1.22	20
1	1.17	1.23	60
2	1.17	1.23	60
4	1.17	1.23	60
8	1.18	1.22	40
16	1.19	1.23	40
32	1.21	1.24	30
64	1.23	1.26	30
128	1.31	1.32	10
256	1.64	1.63	-10
512	1.76	1.70	-60
1024	2.00	1.83	-170
2048	2.88	2.67	-210
4096	3.52	3.44	-80
8192	5.00	4.69	-310

Table 2: OSU Latency Benchmark: Latency between two nodes with and without hardware tag-matching enabled. The fourth column is the difference between baseline and the offloaded, HW tag-matching. Negative values indicate HW TM has lower latency and positive numbers show higher latency. For messages 256 bytes to 8 KiB, HW TM reduces latency.

Loop time is bound with start and stop thresholds, when these minimum thresholds are not met prior to loop time increasing at the rate of work time, this can result in negative values. For this study, we consider the negative values to be zero and report NA. The results shown in Table 3 demonstrate some interesting impacts from rendezvous offloading. For the case of sends, the results indicate a slight increase in overhead when using tag-matching. However, for the receive case, the overhead is entirely offloaded to the HCA. The results also show very low overheads for the base PAMI send, with the lowest measurements not triggering the minimum thresholds. The results also indicate significant overhead in the default PAMI receive results. Further analysis shows that by default PAMI is not progressing posted receives until the application calls into Wait. The third set of results are taken using the PAMI asynchronous background thread. The use of the asynchronous progress thread shows significant reductions in overhead on the receive case, however this comes at the cost of using an additional core per MPI rank.

4.3.2 Miniapps. We followed up our evaluation of hardware tag-matching with applications from the CORAL benchmark suite. In these tests, we evaluated AMG, UMT, and HACC. We selected this set of benchmarks for their mixture of message sizes, with UMT and HACC skewing toward larger messages and AMG with a mix of smaller messages. Spectrum MPI was used to evaluate performance impact which show that hardware tag-matching had little impact or even a slight regression in performance.

Test Case	Min Overhead (μs)	Max Overhead (μs)
Base PAMI Send	NA	7.684
Base PAMI Recv	366.531	376.936
Rendezvous Offload Send	NA	30.966
Rendezvous Offload Recv	NA	10.217
Async PAMI Send	1.941	16.808
Async PAMI Recv	8.099	19.152

Table 3: MPI_Overhead shows significant reductions to the overhead costs of asynchronous receives using rendezvous offloading.

Benchmark	Nodes/PPN	Impact
AMG	192/6	Small Regression
HACC	128/6	No Impact
UMT	192/6	No Impact

Table 4: The impact of using hardware tag-matching on three applications.

4.4 Congestion Management

4.4.1 Micro-benchmarks. As HPC network providers struggle to reduce point-to-point latency below ~600 ns, they are focusing their attention on reducing message tail latency. For collective operations, no rank can make forward progress until the last rank enters the collective. Previous work [23] has identified many sources of system jitter including OS services and processor variation. System jitter causes some ranks to delay and hinder application progression. Network congestion can aggravate this situation when a process' collective packet encounters congested switch queues caused by another application. For example, one applications I/O traffic blocking progress of another applications collective messages.

Cray has developed a pair of new benchmarks to measure network tail latencies. The *network_test* and *network_load_test* benchmarks, which comprise the Global Performance and Congestion Network Test (GPCNeT) [8], provide both mean and 99th percentile measurements for common communication patterns. To minimize the benefits of optimal scheduler placement of ranks contiguously within local groups (e.g., nodes connected to the same TOR in a fat-tree or within the same local group in a Dragonfly), the application will build a random ring similar to the random ring pattern in HPCC[24] such that all communication occurs over the network (i.e., no intra-node communication). This pattern is pessimistic and should provide an upper bound on latency and a lower bound on bandwidth.

Network_test attempts to characterize performance of an application when run in isolation. Ideally, this would be the only application running on the entire system. This test measures five communication patterns sequentially. These communication patterns include: 2-Sided Random Ring (RR) 8 byte Latency, 2-Sided RR 128 KiB Bandwidth, 2-Sided RR 128 KiB Bandwidth with Barrier, Allreduce 8 byte Latency, and All-to-all 128 byte Bandwidth.

These tests do not generate congestion and are meant to characterize performance of an application running in isolation. This test can be used to measure the entire system as well as local groups within the topology (e.g., subtrees in a fat-tree, a local group in a Dragonfly, or a single, all-to-all dimension in a Hyper-X).

Network_load_test, on the other hand, intentionally induces congestion. It builds random rings for the various communication patterns. The "victim" workload will use 20% of the nodes in the job while the "congestors" use the remaining 80%. Each congestor has a unique random ring and the communication patterns include: 2-sided Incast, 2-sided Broadcast, 1-sided Incast, 1-sided Broadcast, and All-to-all. Cray has identified these patterns as the worst "bad actors" within their previous networks. The victim ring runs a subset of the communications from the isolated tests including: 2-sided Random Ring 8 byte Latency, 2-sided Random Ring 128 KiB Bandwidth with Barrier, and Allreduce 8 byte Latency.

Each test provides a simple ratio for each victim communication pattern in which lower is better. For latency measurements, the test will report the ratio of congested latency divided by uncongested latency. For bandwidth measurements, it reports the uncongested bandwidth divided by the congested bandwidth. The tests provide the ratios for both the mean and the 99% values. The ratios are useful as a guide to users to understand how much worse the network can behave when experiencing congestion.

The ratios, by themselves, may be less useful to compare systems with different interconnects or to help guide system architects to select an interconnect. Fortunately, the tests also report the absolute numbers for the mean and 99th percentiles when the victim is run without and with congestion. This avoids the false comparison of an interconnect with a low ratio because the isolated performance is very poor such that the congested performance is not much worse.

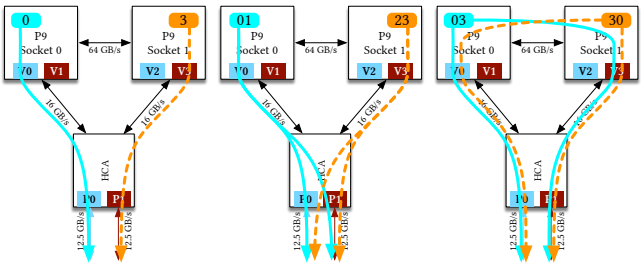

Figure 7: The left images shows the default policy with no striping, which we call 03. The middle image shows striping over PCIe, which we call 0123. The right images shows striping over the SMP bus, which we call 0330.

Figure 7 shows three possible policies for processes to access the network. The current default policy does not stripe (left diagram in Figure 7), thus socket 0 communicates over virtual port 0 and socket 1 communicates over virtual port 3. Spectrum MPI enables striping for messages over 64 KiB, where socket 0 uses virtual ports 0 and 1 and socket 1 uses virtual ports 2 and 3. In this case, eager messages from both sockets only use physical port 0. The user can specify additional port policies and we selected to evaluate striping over the SMP bus where socket 0 uses virtual ports 0 and 3 and socket 1 uses virtual ports 3 and 0. Reversing the order ensures that

socket 1's eager messages use physical port 1 instead of physical port 0.

The network_load_test was used to evaluate the robustness of the three striping policies. The 03 and 0330 policies were found to be superior to the 0123 policy. The 0123 striping policy had significantly higher small message latency and significantly worse 1 MiB bandwidth - especially for the 99% mark. We believe one reason that 0123 policy performs poorly is that all eager messages (≤64 KiB) from both sockets only use physical port 0 and no messages use physical port 1. There may be additional contributors, but we have not had time to investigate yet.

Next, we ran the congestion benchmark ten times on 4,500 nodes of Summit and on 4,200 nodes of Sierra to compare the default (03) to striping over the SMP bus (0330) with AR enabled. On both machines, we used six processes per node. The test reports mean and 99% values for three tests and we present the two-sided 8 byte latency results in detail due to space limitations. The first two tables focus on the mean performance in isolation and when congested. Table 5 shows statistics for the isolated test and there is no difference between the port policies with both displaying very stable results. Table 6 shows the congested results. These are less stable. Compared to Table 5, the defaults mean is 20.7-39X higher while the striping mean is only 7.2-8.8X higher. Clearly, striping provides lower average latencies. With the default port policy, Sierra's tapered network suffers more when congested but striping reduces the impact of the taper.

Isolated Mean Lat (μs)	Summit Defaults (03)	Summit Striping (0330)	Sierra Defaults (03)	Sierra Striping (0330)
Min	2.8	2.8	3.0	3.0
Median	2.8	2.8	3.0	3.0
Mean	2.9	2.8	3.0	3.0
Max	3.8	2.9	3.0	3.0
Std Dev	0.2	0.0	0.0	0.0

Table 5: Statistics for the mean isolated latencies which are very stable. When run in isolation, there is no difference between the defaults and striping over 0330 on either machine.

Table 7 and Table 8 focus on the tail latencies by reporting the 99th percentile values when run in isolation and then under congestion. Table 7 also shows very stable values when run in isolation with little difference between the two policies.[2] Table 8 shows the results for the congested runs. It has the highest standard deviations and the maximum values are 2-4X greater than the minimum values. Again, the striping policy is much better than Summit's current defaults. Compared to Table 7, the defaults policy is 120.1X worse and the striping policy is only 55.9X worse on Summit. Sierra's defaults policy is 206.2X worse and the striping policy is only 54.8X worse. Again, Sierra's taper exacerbates the default policy performance and striping helps reduce the impact.

We caution against using a single sample to determine system policies (e.g., enable AR, select default ports) or to compare between systems with different interconnects. Given the very high standard deviations during congested runs, one cannot rely on a single sample. On these networks, however, single samples can provide a good measure of mean and 99% latencies when run in isolation.

Lastly, we used network_load_test to see the impact of AR when congested. We used the default policy of 03 and ran it 20 times on Summit. Table 9 counter-intuitively shows a large, negative impact on latency with AR is enabled. Tail latencies for the eight-byte message are ~75% worse with AR enabled. We surmise that the switches are detecting congestion and adaptively routing the congestors' traffic that then consume more buffers within the network and

[2]Summit had one outlier in the defaults results which slightly skewed its statistics.

Congested Mean Lat (μs)	Summit Defaults (03)	Summit Striping (0330)	Sierra Defaults (03)	Sierra Striping (0330)
Min	24.3	12.4	63.1	9.8
Median	56.2	18.5	116.9	26.1
Mean	59.9	20.9	117.0	26.3
Max	111.6	63.7	166.5	42.6
Std Dev	21.9	10.8	26.7	9.2

Table 6: Table 5 was in isolation. These results are the mean latencies when the congestors are running at the same time. These congested mean latencies are less stable (i.e., higher standard deviations) than the isolated means. Note, the striping over 0330 is much better (lower) than the defaults.

Isolated 99% Lat (μs)	Summit Defaults (03)	Summit Striping (0330)	Sierra Defaults (03)	Sierra Striping (0330)
Min	6.6	6.5	7.1	7.1
Median	6.6	6.6	7.1	7.1
Mean	8.3	6.7	7.1	7.1
Max	38.1	7.2	7.1	7.1
Std Dev	7.0	0.2	0.0	0.0

Table 7: Compared to Table 5's mean values, these are the isolated 99% values and are also very stable. These represent the tail latency and are 2.3-2.8X the mean isolated values.

Congested 99% Lat (μs)	Summit Defaults (03)	Summit Striping (0330)	Sierra Defaults (03)	Sierra Striping (0330)
Min	621.0	257.0	1198.8	182.6
Median	1003.1	349.9	1424.2	406.9
Mean	996.6	374.3	1464.0	388.8
Max	1209.4	1025.3	2111.7	519.5
Std Dev	160.3	154.3	261.3	93.2

Table 8: When congested, Summit's 99% latencies are 120.1X and 55.9X higher than the isolated 99% latencies in Table 7 and Sierra's 99% latencies are 206.2X and 54.8X higher than the isolated 99% latencies. Again, striping over 0330 is much better than the defaults.

Congested 99% 2-Sided 8B Latency (μs)	Defaults Without AR (03)	Defaults With AR (03)
Min	141.9	289.3
Median	204.0	377.1
Mean	212.3	374.8
Max	310.8	427.1
Std Dev	41.1	38.5

Table 9: When Summit is congested, the 99% latencies for 8 byte 2-sided messages are ~75% higher with AR enabled.

increase the latency observed by the latency-sensitive application. Even though latency is worse with AR during congestion, Table 10 still shows that bandwidth is much better with AR.

Congested 99% 2-Sided 1 MB BW (MB/s)	Defaults Without AR (03)	Defaults With AR (03)
Min	1610.9	2107.3
Median	1802.3	2430.8
Mean	1801.5	2379.4
Max	1914.8	2611.0
Std Dev	82.4	180.1

Table 10: When congested, the 99% bandwidth for 1 MiB 2-sided messages are ~30% higher with AR enabled.

5 EARLY APPLICATION EXPERIENCES

During the open science period, we ran multiple applications on these machines. Because the computation was expected to benefit greatly from the acceleration, we were concerned that message passing would become a bottleneck, and thus took a careful look at the amount of time the codes were spending communicating. In this section, we discuss two applications that ran on both machines and the impacts of tapering. Additionally, we describe a third application's experience with messaging on Sierra and discuss the next steps in addressing the issues of packing and unpacking of messages.

5.1 QUDA

QUDA[1, 6] is a framework that supports the construction of highly-optimized Quantum Chromodynamics (QCD) algorithms for use with NVIDIA GPUs. QCD is the fundamental theory of nuclear strong interactions that governs the formation and structure of protons, neutrons, and atomic nuclei. Understanding basic properties of matter directly from QCD requires the use of HPC. One of the early science applications on Sierra used QCD computations to make the most precise theoretical prediction of the neutron lifetime, a quantity of current scrutiny in the search for new physics. The software development for this effort was recognized as a 2018 Gordon Bell Finalist[3] and the science research is being followed up on both Lassen and Summit.

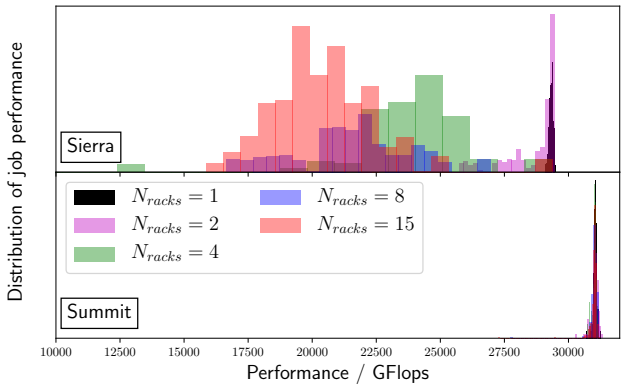

Figure 8: QUDA performance distribution of 4-node jobs on Sierra and Summit versus the number of racks allowed to the job (each rack has 18 nodes). The jobs were run simultaneously in bundles of 4 (1 rack), 8 (2 racks), 16 (4 racks), 32 (8 racks) and 64 (15 racks).

The majority of QCD computational requirements is the solution to large, sparse, linear system solves of the discretized Dirac operator, which must be performed hundreds of thousands of times over a Monte Carlo history. Since QCD is a four-dimensional theory (time plus three space), the stencil kernels result in a lot of nearest-neighbor communication and hence, much off-node memory traffic, making it particularly bandwidth bound compared with other HPC applications. Each of the many computations described above required between one and four nodes of Sierra or Summit, depending on the problem size. Figure 8 shows the distribution performance of sample tasks performed on both machines, 4-node jobs utilizing all GPUs in both cases. The left axis represents the number of samples measured at a particular performance (x-axis) based on rack distribution. To reduce the number of jobs submitted to the queue, tasks were bundled together and run simultaneously with METAQ[2]. On Sierra, we observed a significant performance degradation when the job size was increased to run more tasks simultaneously. The cause was individual tasks having their nodes split between multiple racks, the probability of which increased with increasing job size (larger number of allowed racks). This is observed by the broadening distribution of performance (and lower mean) in the top panel of Figure 8. When the jobs were not allowed to go off a single rack, the performance distribution is tightly peaked near the expected maximum. On Summit, with the 1:1 "non-blocking" fat-tree, we did not observe such performance degradation, with all distributions staying tightly peaked near the expected maximum. Understanding the cause of the performance loss on Sierra (and Lassen) also suggests a simple solution: the bundled jobs just need to be submitted with a restriction on the number of racks they are allowed to use.

5.2 SW4

The large scale runs of SW4[29], a 3-D seismic modeling code, indicate that any effects of the tapering were offset by other factors such as contention for the NIC and NVLink. In SW4, a majority of

the message passing time is spent in the halo exchange involving the packing and unpacking of pinned host memory buffers by GPU kernels or host code followed by MPI_Isends and MPI_Irecvs on the host. We ran the same 7,200 rank problem on 7,200 GPUs on both Sierra and Summit using 1,800 and 1,200 nodes respectively.

The Sierra run took 493 seconds and the Summit run took 509 seconds for a 3% slowdown on Summit. On both machines compute time in GPU kernels was identical so the performance differences were purely communication driven. NVlink data transfers were 30 seconds on Sierra and 40 seconds on Summit for a 33% slowdown on Summit. MPI time on Sierra was 37 seconds while it was 43 seconds on Summit for a 16% overall slowdown. Broken down into its on node MPI_Isends and Irecvs component messages on Summit took 33% longer to leave the node. For both Sending of messages and NVLink transfers, Sierra has 50% more bandwidth per GPU than Summit. Other overheads should be similar on each machine, e.g. the MPI runtime, kernel launch, etc. contribute to a performance gap that is smaller than the difference in raw performance.

With messages taking 33% longer to leave Summit nodes, but performance only 16% slower this implies that the penalty of tapering the Sierra network could be up to 17% of the MPI time. There are other factors that likely make this a bit smaller, e.g. a higher percentage of Summit communication traffic staying within a node. However, the overall impact of the taper on performance is small as MPI takes less than 10% of the overall runtime. Therefore, the overall performance impact is less than 2% of runtime and tapering is a net gain for SW4 throughput because it allows procuring 6-8% more nodes.

5.3 ARES

ARES [9] is a massively parallel, multi-dimensional, multi-physics simulation that used a significant fraction of the early science time on Sierra running Rayleigh-Taylor instability calculations [34]. Capable of running on millions of processors [28], ARES is also portable across architectures [31]. To make a direct comparison between Sierra and a LLNL Broadwell CPU-based cluster, we study a Rayleigh-Taylor problem with 192 million zones, which is suitable for execution on 32 nodes on both architectures. ARES achieves a 13x speedup on Sierra over the Broadwell cluster. For this problem, ARES spends 30% of its runtime in communication routines on Sierra, nearly a 10x increase over the 3.4% of runtime on the Broadwell cluster. Because ARES weak scales well, we saw the same runtime breakdown on the full-machine run on Sierra. While the interconnect bandwidth on Sierra is only 2x higher than the bandwidth of the Broadwell cluster, we weren't expecting the 10x degradation in communication costs.

To ease understanding of the communication cost, we developed Comb [4], a miniapp which represents the Halo exchange in ARES (90% of the communication cost in ARES). Since the bulk of ARES computation is performed on GPUs, application data is located in device memory at the time of communication. Communication routines on Sierra are different than on a CPU-only architecture in two main ways:

(1) Host or device can pack boundary information into buffers;
(2) The buffers can be stored in host or device memory, possibly requiring additional data transfers between the host and the device prior and after the MPI communication.

Figure 9 shows that a significant portion of the communication routine in Comb is spent in message packing and unpacking, and only half of the time is spent in MPI. These results indicate that going forward, ARES will spend a manageable 15% of its time in MPI, easing the impact of relative network bandwidth loss.

The experience with ARES presents a new opportunity for investigating network features on Summit and Sierra that can reduce the cost of packing and unpacking buffers for transmission over the network. One such addition is User-Mode Memory Registration (UMR), the ability to register non-contiguous memory regions for transmission over the network. The mechanism for integration of MPI and UMR is still under investigation, however, this technology may provide the opportunity to reduce or eliminate the need to pack buffers for data transmission eliminating a large cost to many GPU enabled applications.

6 CONCLUSION

The design of the Sierra and Summit networks incorporates new technologies tailored specifically to accelerate HPC workloads. We conclude with a discussion of our measurements and findings related to these new technologies from both users' and facilities' perspectives. We also discuss lessons learned from procurement and working with vendors on advanced features.

6.1 Adaptive Routing

Adaptive routing shows tremendous benefit in bandwidth measurements on both Sierra and Summit. The results indicate a more consistent state of bandwidth available to applications with little drawback under most conditions. AR is transparent to applications, enabled by setting a PAMI environment variable. We recommend that users should enable AR. If a user should choose not to use AR it may create unnecessary congestion in the network impacting their own and other applications running at the same time.

From the facility perspective, AR has some challenges. AR calculations add additional stress to UFM appliances and, on very large networks, network flaps can generate an almost constant recalculation cycle. AR has also shown to be beneficial in diagnosing faulty hardware. The reduction in variability of measurements in tools like MPIGraph made it very easy to identify underperforming hardware leading to replacements during system bring up. IBM has since added this test to the diagnostic tool suite.

6.2 Hardware Tag Matching

Hardware tag-matching and, in particular, rendezvous offloading are promising features built into ConnectX-5. For small, eager messages, HW TM varied from 3% slower for 8 byte messages to 6% faster for 8 KiB messages when compared to Spectrum MPI's software implementation. Users with applications that send a lot of eager messages will need to understand their message sizes to determine when HW TM makes sense.

For users with applications sending larger messages that use a rendezvous protocol (i.e., larger than 64 KiB), HW TM allows for complete overlap without consuming another core for polling or

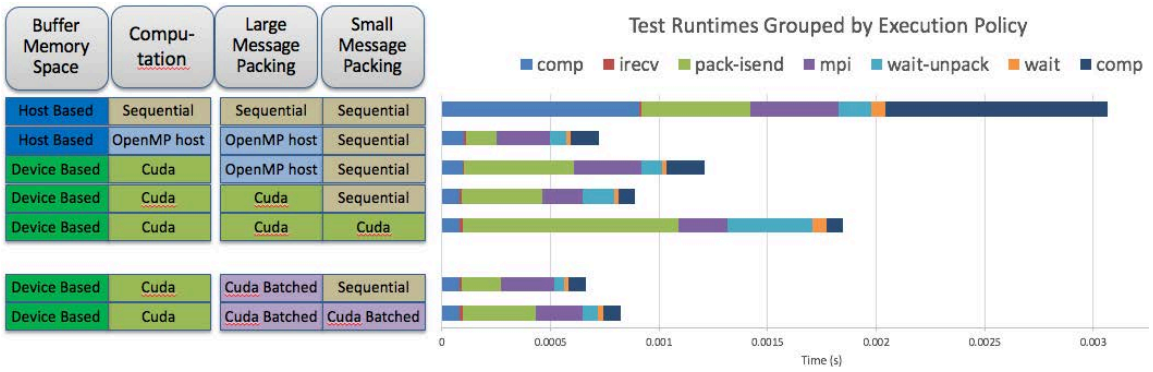

Figure 9: Comb data showing the relative cost of various parts of the communication routine on Sierra

without incurring higher latency for a progress thread that blocks. While Spectrum MPI (and Open-MPI on which it is based) can provide background progress of rendezvous messages when the user selects the async progress thread, this thread consumes a core which the application might be able to use for computation. Rendezvous offloading via HW TM allows the application to use that core for computation. Spectrum MPI, unfortunately, does not allow the user to select a threshold for using HW TM - it is for all two-sided messages or none.

6.3 SHARP

SHARP shows tremendous reduction in the cost of collective algorithms particularly at scale. This has been measured in benchmarks from previous work and as part of the acceptance processes for the machines. However, a deeper exploration of SHARP in applications Nekbone and AMG show that the benefits of SHARP do not necessarily translate to a performance improvement in applications. In many cases, SHARP leads to more variation measured from run-to-run without resulting in overall performance improvements as demonstrated in the AMG results. SHARP can improve application performance as one of the 2018 Gordon Bell Finalists [19] used SHARP in their application and saw improvements in overall kernel performance. This application was heavily dominated by Allreduce collective calls. Feedback to users interested in this feature would be to profile and to consider SHARP only when the application runtime is significantly affected by collective communication. SHARP is still under development and we expect improvements to reduce overheads and for it to benefit more application codes over time.

6.4 Congestion and Tail Latency

We tested multiple policies (e.g., AR, striping, port configurations) on both Summit and Sierra. The congestion benchmark highlights the impact of the tapered network which shows 1.5-2X higher latencies with the current default policy. We identified a promising, albeit non-intuitive, option which stripes data over the SMP bus which brings Sierra's tapered performance closer to Summit's performance. The benchmark, however, avoids intra-node communication unlike real applications that attempt to maximize intra-node communication. We plan on working with real applications to determine if consuming up to 6 GB/s of the 64 GB/s SMP link, in each direction, negatively impacts on-node MPI communication. If not, we will recommend changing the defaults to this configuration.

We also used this benchmark to look at the impact of enabling AR on latency and bandwidth under congestion. Even though AR adds significantly to latency under heavy congestion, the benefits to bandwidth are equally dramatic. When not congested, the latencies are similar with and without AR. We will continue to recommend using AR for all MPI communication.

6.5 Tapering

Tapering the Sierra network resulted in a 6-8% increase in the number of compute nodes given a fixed procurement budget. Data from early Sierra science applications indicates little to no impact on more traditional NNSA workloads (SW4), while the performance of some science workloads such as QCD and some graph algorithms are negatively impacted by the 2:1 network tapering. The former applications have network characteristics that are more representative of the expected workloads on Sierra over its production lifetime, therefore, tapering the network is still justified for Sierra. Regardless, LLNL plans to monitor the performance of Sierra applications to better understand the full impacts of the tapered network.

The science applications (QCD and graph algorithms) impacted by the taper are representative of major workloads expected on Summit and the LLNL unclassified Sierra-like machine, Lassen. Thus, the OLCF's decision to use a non-blocking network is also justified given their workload that includes some FFT heavy codes. At LLNL, more data gathering and analysis is needed to better understand the workloads on the unclassified machines. Data collected on smaller machines was projected forward and this worked well for Sierra when the workloads are similar. However, this projection approach may not be appropriate if the workloads vary between large and small systems.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. DOE by Oak Ridge Leadership Computing Facility at ORNL under contract DE-AC05-00OR22725. The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of

this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Prepared by LLNL under Contract DE-AC52-07NA27344. LLNL-CONF-772398.

REFERENCES

[1] R. Babich, M. A. Clark, B. Joo, G. Shi, R. C. Brower, and S. Gottlieb. 2011. Scaling Lattice QCD beyond 100 GPUs. In *SC11 International Conference for High Performance Computing, Networking, Storage and Analysis Seattle, Washington, November 12-18, 2011*. https://doi.org/10.1145/2063384.2063478 arXiv:hep-lat/1109.2935

[2] Evan Berkowitz. 2017. METAQ: Bundle Supercomputing Tasks. (2017). arXiv:physics.comp-ph/1702.06122

[3] Evan Berkowitz, M. A. Clark, Arjun Gambhir, Ken McElvain, Amy Nicholson, Enrico Rinaldi, Pavlos Vranas, André Walker-Loud, Chia Cheng Chang, Bálint Joó, Thorsten Kurth, and Kostas Orginos. 2018. Simulating the Weak Death of the Neutron in a Femtoscale Universe with Near-exascale Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 55, 9 pages. http://dl.acm.org/citation.cfm?id=3291656.3291730

[4] Jason Burmark. 2019. https://github.com/LLNL/Comb. (2019).

[5] S. Chakraborty, M. Bayatpour, J. Hashmi, H. Subramoni, and D. K. Panda. 2018. Cooperative Rendezvous Protocols for Improved Performance and Overlap. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 28, 13 pages. http://dl.acm.org/citation.cfm?id=3291656.3291694

[6] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi. 2010. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. *Comput. Phys. Commun.* 181 (2010), 1517–1528. https://doi.org/10.1016/j.cpc.2010.05.002 arXiv:hep-lat/0911.3191

[7] Charles Clos. 1953. A study of non-blocking switching networks. *The Bell System Technical Journal* Volume 32, Issue 2 (Mar 1953), 406–424. https://doi.org/10.1002/j.1538-7305.1953.tb01433.x

[8] Cray. 2019. Global Performance and Congestion Network Test - GPCNeT. (2019). https://xgitlab.cels.anl.gov/networkbench/GPCNET

[9] R. Darlington, T. McAbee, and G. Rodrigue. 2001. A Study of ALE Simulations of Rayleigh-Taylor Instability. In *Computer Physics Communications*, Vol. 135. 58–73.

[10] Sylvain Didelot, Patrick Carribault, Marc Pérache, and William Jalby. 2014. Improving MPI Communication Overlap with Collaborative Polling. *Computing* 96, 4 (April 2014), 263–278. https://doi.org/10.1007/s00607-013-0327-z

[11] Douglas Doerfler and Ron Brightwell. 2006. Measuring MPI Send and Receive Overhead and Application Availability in High Performance Network Interfaces. In *Proceedings of the 13th European PVM/MPI User's Group Conference on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI'06)*. Springer-Verlag, Berlin, Heidelberg, 331–338. https://doi.org/10.1007/11846802_46

[12] Jack Dongarra, Hans Meuer, and Erich Strohmaier. 2015. Top500 Supercomputing Sites. http://www.top500.org. (2015).

[13] P. Fischer, J. Lottes, D. Pointer, and A. Siegel. 2008. Petascale algorithms for reactor hydrodynamics. In *Journal of Physics Conference Series (Journal of Physics Conference Series)*, Vol. 125. Article 012076, 012076 pages. https://doi.org/10.1088/1742-6596/125/1/012076

[14] Message P Forum. 1994. *MPI: A Message-Passing Interface Standard.* Technical Report. Knoxville, TN, USA.

[15] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, L. Levi, A. Margolin, T. Ronen, A. Shpiner, O. Wertheim, and E. Zahavi. 2016. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. 1–10. https://doi.org/10.1109/COMHPC.2016.006

[16] Van Emden Henson and Ulrike Meier Yang. 2002. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics* 41, 1 (2002), 155 – 177. https://doi.org/10.1016/S0168-9274(01)00115-5 Developments and Trends in Iterative Methods for Large Systems of Equations - in memorium Rudiger Weiss.

[17] IBM. 2018. IBM Power System AC922 Introduction and Technical Overview. (2018). http://www.redbooks.ibm.com/abstracts/redp5472.html

[18] IBM. 2019. IBM Spectrum MPI. (2019). https://www.ibm.com/us-en/marketplace/spectrum-mpi

[19] Tsuyoshi Ichimura, Kohei Fujita, Takuma Yamaguchi, Akira Naruse, Jack C. Wells, Thomas C. Schulthess, Tjerk P. Straatsma, Christopher J. Zimmer, Maxime Martinasso, Kengo Nakajima, Muneo Hori, and Lalith Maddegedara. 2018. A Fast Scalable Implicit Solver for Nonlinear Time-evolution Earthquake City Problem on Low-ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 49, 11 pages. http://dl.acm.org/citation.cfm?id=3291656.3291722

[20] S. Kumar, Y. Sun, and L. V. KalÃŀ. 2013. Acceleration of an Asynchronous Message Driven Programming Paradigm on IBM Blue Gene/Q. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 689–699. https://doi.org/10.1109/IPDPS.2013.83

[21] Edgar A. León, Ian Karlin, Abhinav Bhatele, Steven H. Langer, Chris Chambreau, Louis H. Howell, Trent D'Hooge, and Matthew L. Leininger. 2016. Characterizing Parallel Scientific Applications on Commodity Clusters: An Empirical Study of a Tapered Fat-tree. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16)*. IEEE Press, Piscataway, NJ, USA, Article 78, 12 pages. http://dl.acm.org/citation.cfm?id=3014904.3015009

[22] Edgar A. León, Ian Karlin, Abhinav Bhatele, Steven H. Langer, Chris Chambreau, Louis H. Howell, Trent D'Hooge, and Matthew L. Leininger. 2016. Characterizing Parallel Scientific Applications on Commodity Clusters: An Empirical Study of a Tapered Fat-tree. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16)*. IEEE Press, Piscataway, NJ, USA, Article 78, 12 pages. http://dl.acm.org/citation.cfm?id=3014904.3015009

[23] Edgar A. LeÃşn, Ian Karlin, and Adam T. Moody. 2016. System Noise Revisited: Enabling Application Scalability and Reproducibility with SMT. In *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. https://doi.org/10.1109/IPDPS.2016.48

[24] Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi. 2006. The HPC Challenge (HPCC) Benchmark Suite. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06)*. ACM, New York, NY, USA, Article 213. https://doi.org/10.1145/1188455.1188677

[25] Mellanox. 2018. Mellanox Adaptive Routing. (2018). https://community.mellanox.com/s/article/howto-configure-adaptive-routing-and-shiel

[26] Mellnox. 2019. (2019). The following environment variables were suggested by Mellanox for SHARP runs: PAMI_IBV_MTU=4096 HCOLL_MAIN_IB=mlx5_0:1 PAMI_IBV_DEBUG_CQE=1 HCOLL_SHARP_NP=2 HCOLL_ENABLE_SHARP=3 HCOLL_ML_USE_SHMSEG_ALLREDUCE=1 SMPI_HCOLL_ENABLE_BCAST=1 PAMI_ENABLE_STRIPING=0 HCOLL_BCOL_P2P_ALLREDUCE_SHARP_MAX=2048 SHARP_COLL_JOB_QUOTA_MAX_GROUPS=4 MLX5_CQE_SIZE=128 SHARP_COLL_JOB_QUOTA_OSTS=64 PAMI_IBV_ADAPTER_AFFINITY=1 HCOLL_ML_USE_SHMSEG_BARRIER=1 HCOLL_ML_DISABLE_ALLREDUCE=0 HCOLL_ML_DISABLE_BCAST=0 PAMID_EAGER_LOCAL=8192 SHARP_COLL_ENABLE_MCAST_TARGET=1 SHARP_COLL_JOB_QUOTA_PAYLOAD_PER_OST=256 PAMI_IBV_CQEDEPTH=4096 PAMI_IBV_OPT_LATENCY=1 PAMI_PMIX_DATACACHE=1 PAMI_IBV_ENABLE_DCT=1 and the following options were passed to –smpiargs: -mca coll_hcoll_enable 1 -mca coll_hcoll_np 0 -mca coll ^basic -mca coll ^ibm -HCOLL -FCA.

[27] Adam Moody. 2009. *Contention-Free Routing for Shift-based Communication in MPI Applications on Large-scale InfiniBand Clusters.* Technical Report. LLNL-TR-418522, Lawrence Livermore National Laboratory.(LLNL), Livermore, CA (USA).

[28] B. E. Morgan and J. A. Greenough. 2015. Large-Eddy and Unsteady RANS Simulations of a Shock-Accelerated Heavy Gas Cylinder. In *Shock Waves*.

[29] N. A. Petersson and B. Sjögreen. 2017. *User's guide to SW4, version 2.0.* Technical Report LLNL-SM-741439. Lawrence Livermore National Laboratory. (Source code available from `geodynamics.org/cig`).

[30] Amit Ruhela, Hari Subramoni, Sourav Chakraborty, Mohammadreza Bayatpour, Pouya Kousha, and Dhabaleswar K. Panda. 2018. Efficient Asynchronous Communication Progress for MPI Without Dedicated Resources. In *Proceedings of the 25th European MPI Users' Group Meeting (EuroMPI'18)*. ACM, New York, NY, USA, Article 14, 11 pages. https://doi.org/10.1145/3236367.3236376

[31] B. S. Ryujin. 2015. Performance and Portability in the Ares Multi-Physics Code. In *DOE High Performance Computing Operational Review (HPCOR) on Scientific Software Architecture for Portability and Performance*.

[32] S. Kumar, A. R. Mamidala, D. A. Faraj, B. Smith, M. Blocksome, B. Cernohous, D. Miller, J. Parker, J. Ratterman, P. Heidelberger, D. Chen, B. Steinmacher-Burrow. 2012. PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer. In *IEEE 26th International Parallel and Distributed Processing Symposium*. 763–773.

[33] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Sa']-ferthwaite, and Charles P. Thacker. 1991. Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links. *IEEE Journal on Selected Areas in Communications* 9 (1991).

[34] A. P. Singh, K. Duraisamy, and B. E. Morgan. 2019. *Data-Augmented Modeling of Transition to Turbulence in Rayleigh-Taylor Mixing Layers"*. Technical Report LLNL-TR-767683.

[35] Sudharshan S. Vazhkudai, Bronis R. de Supinski, Arthur S. Bland, Al Geist, James Sexton, Jim Kahle, Christopher J. Zimmer, Scott Atchley, Sarp Oral, Don E. Maxwell, Veronica G. Vergara Larrea, Adam Bertsch, Robin Goldstone, Wayne Joubert, Chris Chambreau, David Appelhans, Robert Blackmore, Ben Casses, George Chochia, Gene Davison, Matthew A. Ezell, Tom Gooding, Elsa Gonsiorowski, Leopold Grinberg, Bill Hanson, Bill Hartner, Ian Karlin, Matthew L. Leininger, Dustin Leverman, Chris Marroquin, Adam Moody, Martin Ohmacht, Ramesh Pankajakshan, Fernando Pizzano, James H. Rogers, Bryan Rosenburg, Drew Schmidt, Mallikarjun Shankar, Feiyi Wang, Py Watson, Bob Walkup, Lance D. Weems, and Junqi Yin. 2018. The Design, Deployment, and Evaluation of the CORAL Pre-exascale Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 52, 12 pages. http://dl.acm.org/citation.cfm?id=3291656.3291726

[36] M. G. Venkata, P. Shamis, R. Sampath, R. L. Graham, and J. S. Ladd. 2013. Optimizing blocking and nonblocking reduction operations for multicore systems: Hierarchical design and implementation. In *2013 IEEE International Conference on Cluster Computing*. IEEE, 1–8. https://doi.org/10.1109/CLUSTER.2013.6702676

[37] Vitali Morozov. [n. d.]. ALCF MPI BENCHMARKS. https://www.alcf.anl.gov/software (visited March 2018). ([n. d.]).

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Summit Tests: OSU Benchmarks: 5.5 Compiled using Spectrum MPI 10.2, XL 16.1 Coral Optimized AMG2013 Compile using PGI 18.7, Spectrum MPI 10.2 Cuda 9.2 Strong Scaling AMG 2013 (Base) XL 16.1 Spectrum MPI 10.2 (HCOLL Integrated) Global Performance and Congestion Network Test - GPCNeT, Spectrum MPI 10.2, XL 16.1 Sierra Tests: Compiler XL v16.1.1 Spectrum MPI 10.2.0.11rtm2 SW4 commit f10323c027c42fac8d63a7312ca520a5f11d656b from: https://github.com/geodynamics/sw4.git The CG benchmark and Nekbone application were run with the following command line parameters: "-E HCOLL_MAIN_IB=mlx5_0:1 -E PAMI_IBV_DEBUG_CQE=1 -E SMPI_HCOLL_ENABLE_BCAST=1 -E HCOLL_ML_USE_SHMSEG_BARRIER=1 -E HCOLL_ML_USE_SHMSEG_ALLREDUCE=1 -E HCOLL_ML_DISABLE_ALLREDUCE=0 -E HCOLL_ML_DISABLE_BCAST=0 -E PAMI_IBV_MTU=4096 -E PAMID_EAGER_LOCAL=8192 -E PAMI_ENABLE_STRIPING=0 -E MLX5_CQE_SIZE=128 -E PAMI_IBV_CQEDEPTH=4096 -E PAMI_IBV_ADAPTER_AFFINITY=1 -E PAMI_IBV_OPT_LATENCY=1 -E PAMI_PMIX_DATACACHE=1 -E PAMI_IBV_ENABLE_DCT=1" SMPIARGS="-mca coll_hcoll_enable 1 -mca coll_hcoll_np 0 -mca coll ^basic -mca coll ^ibm -HCOLL -FCA" When testing SHARP results, the following additional parameters were used: -E HCOLL_ENABLE_SHARP=3 -E HCOLL_SHARP_NP=2 -E SHARP_COLL_LOG_LEVEL=3 -E HCOLL_BCOL_P2P_ALLREDUCE_SHARP_MAX=2048 -E SHARP_COLL_JOB_QUOTA_MAX_GROUPS=4 -E SHARP_COLL_JOB_QUOTA_OSTS=64 -E SHARP_COLL_ENABLE_MCAST_TARGET=1 -E SHARP_COLL_JOB_QUOTA_PAYLOAD_PER_OST=256 Both the CG benchmark and Nekbone were built with XL compilers, Spectrum MPI 10.2 and Nekbone was built with CUDA 9.2. The Nekbone source was that used for the CORAL benchmarks and acceptance testing.

ARTIFACT AVAILABILITY

Software Artifact Availability: There are no author-created software artifacts.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: There are no author-created data artifacts.

Proprietary Artifacts: No author-created artifacts are proprietary.

List of URLs and/or DOIs where artifacts are available:

None

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Summit, Sierra

Operating systems and versions: RHEL 7.4

Compilers and versions: XL 16, PGI 18, gcc7.3.1

Applications and versions: OSU 5.5, AMG2013, QUDA 0.9.0

Libraries and versions: Spectrum MPI 10.2

Output from scripts that gathers execution environment information.

```
% cat env-redacted.out
USER=[redacted]
LOGNAME=[redacted]
HOME=/ccs/home/[redacted]
PATH=/ccs/home/[redacted]/projects/stagesend:/usr/bi⌋
↪ n:/usr/sbin:/sw/sources/lsf-tools/2.0/summit/bin⌋
↪ :/sw/sources/lsf-tools/2.0/summit/bin:/sw/summit⌋
↪ /xalt/1.1.3/bin:/autofs/nccs-svm1_sw/summit/.swc⌋
↪ i/1-compute/opt/spack/20180914/linux-rhel7-ppc64⌋
↪ le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23⌋
↪ vuzrlmenbokfpn5sk6n/bin:/sw/sources/hpss/bin:/au⌋
↪ tofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spa⌋
↪ ck/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spec⌋
↪ trum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcn⌋
↪ b6mllodtyfuz/bin:/sw/summit/xl/16.1.1-1/xlC/16.1⌋
↪ .1/bin:/sw/summit/xl/16.1.1-1/xlf/16.1.1/bin:/op⌋
↪ t/ibm/spectrumcomputing/lsf/10.1/linux3.10-glibc⌋
↪ 2.17-ppc64le-csm/etc:/opt/ibm/spectrumcomputing/⌋
↪ lsf/10.1/linux3.10-glibc2.17-ppc64le-csm/bin:/op⌋
↪ t/ibm/csm/bin:/usr/local/bin:/usr/bin:/ccs/home/⌋
↪ [redacted]/bin:/usr/local/sbin:/usr/sbin:/opt/ib⌋
↪ m/flightlog/bin:/opt/ibutils/bin:/opt/ibm/spectr⌋
↪ um_mpi/jsm_pmix/bin:/opt/puppetlabs/bin:/usr/lpp⌋
↪ /mmfs/bin
MAIL=/var/spool/mail/[redacted]
SHELL=/bin/zsh
SSH_CLIENT=[redacted] [redacted] 22
SSH_CONNECTION=[redacted] [redacted] [redacted] 22
SSH_TTY=/dev/pts/23
TERM=xterm-256color
XDG_SESSION_ID=8831
XDG_RUNTIME_DIR=/run/user/58031
SHLVL=1
PWD=/gpfs/alpine/scratch/[redacted]/[redacted]/cray-⌋
↪ congestion-test-1M/266561
OLDPWD=/gpfs/alpine/[redacted]
HISTCONTROL=ignoredups
HOSTNAME=login5
HISTSIZE=2000
```

```
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:p
↪ i=40;38;5;11:so=38;5;13:do=38;5;5:bd=48;5;232;38
↪ ;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=0
↪ 5;48;5;232;38;5;15:su=48;5;196;38;5;15:sg=48;5;1
↪ 1;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38;5;1
↪ 6:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;
↪ 34:*.tar=38;5;9:*.tgz=38;5;9:*.arc=38;5;9:*.arj=
↪ 38;5;9:*.taz=38;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.
↪ lzh=38;5;9:*.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5
↪ ;9:*.tzo=38;5;9:*.t7z=38;5;9:*.zip=38;5;9:*.z=38
↪ ;5;9:*.Z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38
↪ ;5;9:*.lz=38;5;9:*.lzo=38;5;9:*.xz=38;5;9:*.bz2=
↪ 38;5;9:*.bz=38;5;9:*.tbz=38;5;9:*.tbz2=38;5;9:*.
↪ tz=38;5;9:*.deb=38;5;9:*.rpm=38;5;9:*.jar=38;5;9
↪ :*.war=38;5;9:*.ear=38;5;9:*.sar=38;5;9:*.rar=38
↪ ;5;9:*.alz=38;5;9:*.ace=38;5;9:*.zoo=38;5;9:*.cp
↪ io=38;5;9:*.7z=38;5;9:*.rz=38;5;9:*.cab=38;5;9:*
↪ .jpg=38;5;13:*.jpeg=38;5;13:*.gif=38;5;13:*.bmp=
↪ 38;5;13:*.pbm=38;5;13:*.pgm=38;5;13:*.ppm=38;5;1
↪ 3:*.tga=38;5;13:*.xbm=38;5;13:*.xpm=38;5;13:*.ti
↪ f=38;5;13:*.tiff=38;5;13:*.png=38;5;13:*.svg=38;
↪ 5;13:*.svgz=38;5;13:*.mng=38;5;13:*.pcx=38;5;13:
↪ *.mov=38;5;13:*.mpg=38;5;13:*.mpeg=38;5;13:*.m2v
↪ =38;5;13:*.mkv=38;5;13:*.webm=38;5;13:*.ogm=38;5
↪ ;13:*.mp4=38;5;13:*.m4v=38;5;13:*.mp4v=38;5;13:*
↪ .vob=38;5;13:*.qt=38;5;13:*.nuv=38;5;13:*.wmv=38
↪ ;5;13:*.asf=38;5;13:*.rm=38;5;13:*.rmvb=38;5;13:
↪ *.flc=38;5;13:*.avi=38;5;13:*.fli=38;5;13:*.flv=
↪ 38;5;13:*.gl=38;5;13:*.dl=38;5;13:*.xcf=38;5;13:
↪ *.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=
↪ 38;5;13:*.axv=38;5;13:*.anx=38;5;13:*.ogv=38;5;1
↪ 3:*.ogx=38;5;13:*.aac=38;5;45:*.au=38;5;45:*.fla
↪ c=38;5;45:*.mid=38;5;45:*.midi=38;5;45:*.mka=38;
↪ 5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg=38;5;45:*
↪ .ra=38;5;45:*.wav=38;5;45:*.axa=38;5;45:*.oga=38
↪ ;5;45:*.spx=38;5;45:*.xspf=38;5;45:
CVS_RSH=ssh
LESSOPEN=||/usr/bin/lesspipe.sh %s
BINARY_TYPE_HPC=
LD_LIBRARY_PATH=/autofs/nccs-svm1_sw/summit/.swci/1-
↪ compute/opt/spack/20180914/linux-rhel7-ppc64le/g
↪ cc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzr
↪ lmenbokfpn5sk6n/lib:/autofs/nccs-svm1_sw/summit/
↪ .swci/1-compute/opt/spack/20180914/linux-rhel7-p
↪ pc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-201812
↪ 14-y75ruim4h6cvqcdymmcnb6mllodtyfuz/lib:/sw/summ
↪ it/xl/16.1.1-1/xlsmp/5.1.1/lib:/sw/summit/xl/16.
↪ 1.1-1/xlmass/9.1.1/lib:/sw/summit/xl/16.1.1-1/xl
↪ C/16.1.1/lib:/sw/summit/xl/16.1.1-1/xlf/16.1.1/l
↪ ib:/sw/summit/xl/16.1.1-1/lib:/opt/ibm/spectrumc
↪ omputing/lsf/10.1/linux3.10-glibc2.17-ppc64le-cs
↪ m/lib
LSF_SERVERDIR=/opt/ibm/spectrumcomputing/lsf/10.1/li
↪ nux3.10-glibc2.17-ppc64le-csm/etc
LSF_BINDIR=/opt/ibm/spectrumcomputing/lsf/10.1/linux
↪ 3.10-glibc2.17-ppc64le-csm/bin
LSF_LIBDIR=/opt/ibm/spectrumcomputing/lsf/10.1/linux
↪ 3.10-glibc2.17-ppc64le-csm/lib
XLSF_UIDDIR=/opt/ibm/spectrumcomputing/lsf/10.1/linu
↪ x3.10-glibc2.17-ppc64le-csm/lib/uid
LSF_ENVDIR=/opt/ibm/spectrumcomputing/lsf/conf
MANPATH=/sw/sources/hpss/man:/autofs/nccs-svm1_sw/su
↪ mmit/.swci/1-compute/opt/spack/20180914/linux-rh
↪ el7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-2
↪ 0181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz/share/m
↪ an:/sw/summit/xl/16.1.1-1/xlC/16.1.1/man/en_US:/
↪ sw/summit/xl/16.1.1-1/xlf/16.1.1/man/en_US:/sw/s
↪ ummit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmod/lmod/sha
↪ re/man:/opt/ibm/spectrumcomputing/lsf/10.1/man::
PS1=[%{%}%n%{%}@%{%}%m%{%}]%{%}%3c %{%}%#
OLCF_LMOD_ROOT=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8
↪ .5
LMOD_sys=Linux
MODULEPATH_ROOT=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.
↪ 8.5/modulefiles
MODULEPATH=/autofs/nccs-svm1_sw/summit/modulefiles/s
↪ ite/linux-rhel7-ppc64le/spectrum-mpi/10.2.0.10-2
↪ 0181214-y75ruim/xl/16.1.1-1:/sw/summit/modulefil
↪ es/site/linux-rhel7-ppc64le/xl/16.1.1-1:/sw/summ
↪ it/modulefiles/site/linux-rhel7-ppc64le/Core:/sw
↪ /summit/modulefiles/core:/sw/summit/lmod/7.7.10/
↪ rhel7.3_gnu4.8.5/modulefiles/Linux:/sw/summit/lm
↪ od/7.7.10/rhel7.3_gnu4.8.5/modulefiles/Core:/sw/
↪ summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmod/lmod/mo
↪ dulefiles/Core
BASH_ENV=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod/init/bash
LMOD_PKG=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod
LMOD_DIR=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod/libexec
LMOD_CMD=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod/libexec/lmod
MODULESHOME=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/
↪ lmod/lmod
LMOD_SETTARG_FULL_SUPPORT=no
LMOD_VERSION=7.7.10
SHOST=login5
_ModuleTable001_=X01vZHVsZVRhYmxlXz17WyJNVHZlcnNpb24
↪ iXT0zLFsiY19yZWJ1aWxkVGltZSJdPWZhbHNlLFsiY19zaG9
↪ ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9e30sZmFtaWx5PXtbImN
↪ vbXBpbGVyIl09InhsIixbIm1waSJdPSJzcGVjdHJ1bS1tcGk
↪ iLH0sbVQ9e0RlZkFwcHM9e1siZm4iXT0iL3N3L3N1bW1pdC9
↪ tb2R1bGVmaWxlcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0bGU
↪ vQ29yZS9EZWZBcHBzLmx1YSIsWyJmdWxsTmFtZSJdPSJEZWZ
↪ BcHBzIixbImxvYWRPcmRlciJdPTcscHJvcFQ9e30sWyJzdGF
↪ ja0RlcHRoIl09MCxbInN0YXR1cyJdPSJhY3RpdmUiLFsidXN
↪ lck5hbWUiXT0iRGVmQXBwcyIsfSxbImRhcnNoYW4tcnVudGl
↪ tZSJdPXtbImZuIl09Ii9zdy9zdW1taXQvbW9kdWxlZmls
```

```
_ModuleTable002_=ZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGx
↪ lL0NvcmUvZGFyc2hhbi1ydW50aW1lLzMuMS42Lmx1YSIsWyJ
↪ mdWxsTmFtZSJdPSJkYXJzaGFuLXJ1bnRpbWUvMy4xLjYiLFs
↪ ibG9hZE9yZGVyIl09Nixwcm9wVD17fSxbInN0YWNrRGVwdGg
↪ iXT0xLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJ
↪ dPSJkYXJzaGFuLXJ1bnRpbWUiLH0saHNpPXtbImZuIl09Ii9
↪ zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGV
↪ sNy1wcGM2NGxlL0NvcmUvaHNpLzUuMC4yLnA1Lmx1YSIsWyJ
↪ mdWxsTmFtZSJdPSJoc2kvNS4wLjIucDUiLFsibG9hZE9yZGV
↪ yIl09Myxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0xLFsic3R
↪ hdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJoc2ki
_ModuleTable_Sz_=6
__Init_Default_Modules=1
LMOD_SYSTEM_DEFAULT_MODULES=DefApps
__LMOD_REF_COUNT_CMAKE_PREFIX_PATH=/autofs/nccs-svm1
↪ _sw/summit/.swci/1-compute/opt/spack/20180914/li
↪ nux-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.
↪ 6-wwbm2i5cife23vuzrlmenbokfpn5sk6n:1;/autofs/ncc
↪ s-svm1_sw/summit/.swci/1-compute/opt/spack/20180
↪ 914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi
↪ -10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodty
↪ fuz:1
CMAKE_PREFIX_PATH=/autofs/nccs-svm1_sw/summit/.swci/
↪ 1-compute/opt/spack/20180914/linux-rhel7-ppc64le
↪ /gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vu
↪ zrlmenbokfpn5sk6n:/autofs/nccs-svm1_sw/summit/.s
↪ wci/1-compute/opt/spack/20180914/linux-rhel7-ppc
↪ 64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214
↪ -y75ruim4h6cvqcdymmcnb6mllodtyfuz
__LMOD_REF_COUNT_CPATH=/autofs/nccs-svm1_sw/summit/.
↪ swci/1-compute/opt/spack/20180914/linux-rhel7-pp
↪ c64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-2018121
↪ 4-y75ruim4h6cvqcdymmcnb6mllodtyfuz/include:1
CPATH=/autofs/nccs-svm1_sw/summit/.swci/1-compute/op
↪ t/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1
↪ /spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcd
↪ ymmcnb6mllodtyfuz/include
__LMOD_REF_COUNT_LD_LIBRARY_PATH=/autofs/nccs-svm1_s
↪ w/summit/.swci/1-compute/opt/spack/20180914/linu
↪ x-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-
↪ wwbm2i5cife23vuzrlmenbokfpn5sk6n/lib:1;/autofs/n
↪ ccs-svm1_sw/summit/.swci/1-compute/opt/spack/201
↪ 80914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-m
↪ pi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllo
↪ dtyfuz/lib:1;/sw/summit/xl/16.1.1-1/xlsmp/5.1.1/
↪ lib:1;/sw/summit/xl/16.1.1-1/xlmass/9.1.1/lib:1;
↪ /sw/summit/xl/16.1.1-1/xlC/16.1.1/lib:1;/sw/summ
↪ it/xl/16.1.1-1/xlf/16.1.1/lib:1;/sw/summit/xl/16
↪ .1.1-1/lib:1;/opt/ibm/spectrumcomputing/lsf/10.1
↪ /linux3.10-glibc2.17-ppc64le-csm/lib:1
__LMOD_REF_COUNT_LIBRARY_PATH=/autofs/nccs-svm1_sw/s
↪ ummit/.swci/1-compute/opt/spack/20180914/linux-r
↪ hel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwb
↪ m2i5cife23vuzrlmenbokfpn5sk6n/lib:1;/autofs/nccs
↪ -svm1_sw/summit/.swci/1-compute/opt/spack/2018091
↪ 4/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-1
↪ 0.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyf
↪ uz/lib:1
LIBRARY_PATH=/autofs/nccs-svm1_sw/summit/.swci/1-com
↪ pute/opt/spack/20180914/linux-rhel7-ppc64le/gcc-
↪ 4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzrlme
↪ nbokfpn5sk6n/lib:/autofs/nccs-svm1_sw/summit/.sw
↪ ci/1-compute/opt/spack/20180914/linux-rhel7-ppc6
↪ 4le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-
↪ y75ruim4h6cvqcdymmcnb6mllodtyfuz/lib
LMOD_FAMILY_COMPILER=xl
LMOD_FAMILY_COMPILER_VERSION=16.1.1-1
LMOD_FAMILY_MPI=spectrum-mpi
LMOD_FAMILY_MPI_VERSION=10.2.0.10-20181214
LMOD_MPI_NAME=spectrum-mpi
LMOD_MPI_VERSION=10.2.0.10-20181214-y75ruim
__LMOD_REF_COUNT_LOADEDMODULES=xl/16.1.1-1:1;spectru
↪ m-mpi/10.2.0.10-20181214:1;hsi/5.0.2.p5:1;xalt/1
↪ .1.3:1;lsf-tools/2.0:1;darshan-runtime/3.1.6:1;D
↪ efApps:1
LOADEDMODULES=xl/16.1.1-1:spectrum-mpi/10.2.0.10-201
↪ 81214:hsi/5.0.2.p5:xalt/1.1.3:lsf-tools/2.0:dars
↪ han-runtime/3.1.6:DefApps
__LMOD_REF_COUNT_MANPATH=/sw/sources/hpss/man:1;/aut
↪ ofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spac
↪ k/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spect
↪ rum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb
↪ 6mllodtyfuz/share/man:1;/sw/summit/xl/16.1.1-1/x
↪ lC/16.1.1/man/en_US:1;/sw/summit/xl/16.1.1-1/xlf
↪ /16.1.1/man/en_US:1;/sw/summit/lmod/7.7.10/rhel7
↪ .3_gnu4.8.5/lmod/lmod/share/man:1;/opt/ibm/spect
↪ rumcomputing/lsf/10.1/man:1
MPI_ROOT=/autofs/nccs-svm1_sw/summit/.swci/1-compute
↪ /opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.
↪ 1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cv
↪ qcdymmcnb6mllodtyfuz
__LMOD_REF_COUNT_NLSPATH=/sw/summit/xl/16.1.1-1/msg/
↪ en_US/%N:2;/sw/summit/xl/16.1.1-1/xlC/16.1.1/msg
↪ /en_US/%N:1;/sw/summit/xl/16.1.1-1/xlf/16.1.1/ms
↪ g/en_US/%N:1
NLSPATH=/sw/summit/xl/16.1.1-1/msg/en_US/%N:/sw/summ
↪ it/xl/16.1.1-1/xlC/16.1.1/msg/en_US/%N:/sw/summi
↪ t/xl/16.1.1-1/xlf/16.1.1/msg/en_US/%N
OLCF_DARSHAN_RUNTIME_ROOT=/autofs/nccs-svm1_sw/summi
↪ t/.swci/1-compute/opt/spack/20180914/linux-rhel7
↪ -ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5c
↪ ife23vuzrlmenbokfpn5sk6n
OLCF_HSI_ROOT=/sw/sources/hpss
```

```
OLCF_SPECTRUM_MPI_ROOT=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz
OLCF_XLC_ROOT=/sw/summit/xl/16.1.1-1/xlC/16.1.1
OLCF_XLF_ROOT=/sw/summit/xl/16.1.1-1/xlf/16.1.1
OLCF_XLMASS_ROOT=/sw/summit/xl/16.1.1-1/xlmass/9.1.1
OLCF_XLSMP_ROOT=/sw/summit/xl/16.1.1-1/xlsmp/5.1.1
OLCF_XL_ROOT=/sw/summit/xl/16.1.1-1
OMPI_CC=/sw/summit/xl/16.1.1-1/xlC/16.1.1/bin/xlc_r
OMPI_CXX=/sw/summit/xl/16.1.1-1/xlC/16.1.1/bin/xlc++_r
OMPI_DIR=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz
OMPI_FC=/sw/summit/xl/16.1.1-1/xlf/16.1.1/bin/xlf2008_r
OMPI_LD_PRELOAD_PREPEND=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzrlmenbokfpn5sk6n/lib/libdarshan.so
OPAL_LIBDIR=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz/lib
OPAL_PREFIX=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz
PAMI_IBV_ENABLE_OOO_AR=1
PAMI_IBV_QP_SERVICE_LEVEL=8
__LMOD_Priority_PATH=/sw/sources/lsf-tools/2.0/summit/bin:-9999;/sw/summit/xalt/1.1.3/bin:-9999
__LMOD_REF_COUNT_PATH=/sw/sources/lsf-tools/2.0/summit/bin:1;/sw/summit/xalt/1.1.3/bin:1;/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzrlmenbokfpn5sk6n/bin:1;/sw/sources/hpss/bin:1;/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz/bin:1;/sw/summit/xl/16.1.1-1/xlC/16.1.1/bin:1;/sw/summit/xl/16.1.1-1/xlf/16.1.1/bin:1;/opt/ibm/spectrumcomputing/lsf/10.1/linux3.10-glibc2.17-ppc64le-csm/etc:1;/opt/ibm/spectrumcomputing/lsf/10.1/linux3.10-glibc2.17-ppc64le-csm/bin:1;/opt/ibm/csm/bin:1;/usr/local/bin:1;/usr/bin:1;/ccs/home/[redacted]/bin:1;/usr/local/sbin:1;/usr/sbin:1;/opt/ibm/flightlog/bin:1;/opt/ibutils/bin:1;/opt/ibm/spectrum_mpi/jsm_pmix/bin:1
__LMOD_REF_COUNT_PKG_CONFIG_PATH=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzrlmenbokfpn5sk6n/lib/pkgconfig:1
PKG_CONFIG_PATH=/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzrlmenbokfpn5sk6n/lib/pkgconfig
__LMOD_REF_COUNT_PYTHONPATH=/sw/summit/xalt/1.1.3/site:1;/sw/summit/xalt/1.1.3/libexec:1
PYTHONPATH=/ccs/home/[redacted]/projects/stagesend:/usr/bin:/usr/sbin:/sw/sources/lsf-tools/2.0/summit/bin:/sw/sources/lsf-tools/2.0/summit/bin:/sw/summit/xalt/1.1.3/bin:/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/darshan-runtime-3.1.6-wwbm2i5cife23vuzrlmenbokfpn5sk6n/bin:/sw/sources/hpss/bin:/autofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.10-20181214-y75ruim4h6cvqcdymmcnb6mllodtyfuz/bin:/sw/summit/xl/16.1.1-1/xlC/16.1.1/bin:/sw/summit/xl/16.1.1-1/xlf/16.1.1/bin:/opt/ibm/spectrumcomputing/lsf/10.1/linux3.10-glibc2.17-ppc64le-csm/etc:/opt/ibm/spectrumcomputing/lsf/10.1/linux3.10-glibc2.17-ppc64le-csm/bin:/opt/ibm/csm/bin:/usr/local/bin:/usr/bin:/ccs/home/[redacted]/bin:/usr/local/sbin:/usr/sbin:/opt/ibm/flightlog/bin:/opt/ibutils/bin:/opt/ibm/spectrum_mpi/jsm_pmix/bin:/opt/puppetlabs/bin:/usr/lpp/mmfs/bin:/ccs/home/[redacted]/projects/pexpect-2.3/build/lib:/ccs/home/[redacted]/projects/stagesend
XALT_ETC_DIR=/sw/summit/xalt/1.1.3/etc
XALT_OLCF=1
XL_LINKER=/sw/summit/xalt/1.1.3/bin/ld
__LMOD_REF_COUNT__LMFILES_=/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/xl/16.1.1-1.lua:1;/sw/summit/modulefiles/site/linux-rhel7-ppc64le/xl/16.1.1-1/spectrum-mpi/10.2.0.10-20181214.lua:1;/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/hsi/5.0.2.p5.lua:1;/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/xalt/1.1.3.lua:1;/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/lsf-tools/2.0.lua:1;/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/darshan-runtime/3.1.6.lua:1;/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/DefApps.lua:1
_LMFILES_=/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/xl/16.1.1-1.lua:/sw/summit/modulefiles/site/linux-rhel7-ppc64le/xl/16.1.1-1/spectrum-mpi/10.2.0.10-20181214.lua:/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/hsi/5.0.2.p5.lua:/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/xalt/1.1.3.lua:/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/lsf-tools/2.0.lua:/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/darshan-runtime/3.1.6.lua:/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/DefApps.lua
```

```
_ModuleTable003_=LH0sWyJsc2YtdG9vbHMiXT17WyJmbiJdPSI
↪  vc3cvc3VtbWl0L21vZHVsZWZpbGVzL3NpdGUvbGludXgtcmh
↪  lbDctcHBjNjRsZS9Db3JlL2xzZi10b29scy8yLjAubHVhIix
↪  bImZ1bGxOYW1lIl09ImxzZi10b29scy8yLjAiLFsibG9hZE9
↪  yZGVyIl09NSxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0xLFs
↪  ic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJsc2Y
↪  tdG9vbHMiLH0sWyJzcGVjdHJ1bS1tcGkiXT17WyJmbiJdPSI
↪  vc3cvc3VtbWl0L21vZHVsZWZpbGVzL3NpdGUvbGludXgtcmh
↪  lbDctcHBjNjRsZS94bC8xNi4xLjEtMS9zcGVjdHJ1bS1tcGk
↪  vMTAuMi4wLjEwLTIwMTgxMjE0Lmx1YSIsWyJmdWxsTmFtZSJ
↪  dPSJzcGVjdHJ1bS1tcGkvMTAuMi4wLjEwLTIwMTgxMjE0
_ModuleTable004_=IixbImxvYWRPcmRlciJdPTIscHJvcFQ9e30
↪  sWyJzdGFja0RlcHRoIl09MSxbInN0YXR1cyJdPSJhY3RpdmU
↪  iLFsidXNlck5hbWUiXT0ic3BlY3RydW0tbXBpIix9LHhhbHQ
↪  9e1siZm4iXT0iL3N3L3N1bW1pdC9tb2R1bGVmaWxlcy9zaXR
↪  lL2xpbnV4LXJoZWw3LXBwYzY0bGUvQ29yZS94YWx0LzEuMS4
↪  zLmx1YSIsWyJmdWxsTmFtZSJdPSJ4YWx0LzEuMS4zIixbImx
↪  vYWRPcmRlciJdPTQscHJvcFQ9e30sWyJzdGFja0RlcHRoIl0
↪  9MSxbInN0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5hbWUiXT0
↪  ieGFsdCIsfSx4bD17WyJmbiJdPSIvc3cvc3VtbWl0L21vZHV
↪  sZWZpbGVzL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS9Db3J
↪  lL3hsLzE2LjEuMS0xLmx1YSIsWyJmdWxsTmFtZSJdPSJ4
_ModuleTable005_=bC8xNi4xLjEtMSIsWyJsb2FkT3JkZXIiXT0
↪  xLHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTEsWyJzdGF0dXM
↪  iXT0iYWN0aXZlIixbInVzZXJOYW1lIl09InhsIix9LH0sbXB
↪  hdGhBPXsiL2F1dG9mcy9uY2NzLXN2bTFfc3cvc3VtbWl0L21
↪  vZHVsZWZpbGVzL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS9
↪  zcGVjdHJ1bS1tcGkvMTAuMi4wLjEwLTIwMTgxMjE0LXk3NXJ
↪  1aW0veGwvMTYuMS4xLTEiLCIvc3cvc3VtbWl0L21vZHVsZWZ
↪  pbGVzL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS94bC8xNi4
↪  xLjEtMSIsIi9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9
↪  saW51eC1yaGVsNy1wcGM2NGxlL0NvcmUiLCIvc3cvc3VtbWl
↪  0L21vZHVsZWZpbGVzL2NvcmUiLCIvc3cvc3VtbWl0L2xt
_ModuleTable006_=b2QvNy43LjEwL3JoZWw3LjNfZ251NC44LjU
↪  vbW9kdWxlZmlsZXMvTGludXgiLCIvc3cvc3VtbWl0L2xtb2Q
↪  vNy43LjEwL3JoZWw3LjNfZ251NC44LjUvbW9kdWxlZmlsZXM
↪  vQ29yZSIsIi9zdy9zdW1taXQvbG1vZC83LjcuMTAvcmhlbDc
↪  uM19nbnU0LjguNS9sbW9kL2xtb2QvbW9kdWxlZmlsZXMvQ29
↪  yZSIsfSxbInN5c3RlbUJhc2VNUEFUSCJdPSIvc3cvc3VtbWl
↪  0L2xtb2QvNy43LjEwL3JoZWw3LjNfZ251NC44LjUvbW9kdWx
↪  lZmlsZXMvTGludXg6L3N3L3N1bW1pdC9sbW9kLzcuNy4xMC9
↪  yaGVsNy4zX2dudTQuOC41L21vZHVsZWZpbGVzL0NvcmU6L3N
↪  3L3N1bW1pdC9sbW9kLzcuNy4xMC9yaGVsNy4zX2dudTQuOC4
↪  1L2xtb2QvbG1vZC9tb2R1bGVmaWxlcy9Db3JlIix9
OLCF_MODULEPATH_ROOT=/sw/summit/modulefiles
MEMBERWORK=/gpfs/alpine/scratch/[redacted]
PROJWORK=/gpfs/alpine/proj-shared
WORLDWORK=/gpfs/alpine/world-shared
PAMI_ENABLE_STRIPING=0
PAMI_IBV_ENABLE_DCT=1
PAMI_IBV_ADAPTER_AFFINITY=1
OMPI_MCA_io=romio314
OMPI_MCA_coll_ibm_xml_disable_cache=1
PAMI_PMIX_USE_OLD_MAPCACHE=1
LESS=-RseFiX
EDITOR=/usr/bin/vim
SAVEHIST=1000
HISTFILE=/ccs/home/[redacted]/.zsh/zsh_history
_=/usr/bin/env


% module list

Currently Loaded Modules:
  1) xl/16.1.1-3                      3) hsi/5.0.2.p5
  ↪  5) lsf-tools/2.0           7) DefApps
  2) spectrum-mpi/10.3.0.0-20190419   4) xalt/1.1.3
  ↪   6) darshan-runtime/3.1.7
```