GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism

Kiran Ravikumar
kiran.r@gatech.edu
Georgia Institute of Technology
Atlanta, GA

David Appelhans
dappelh@us.ibm.com
IBM Research
Boulder, CO

P.K. Yeung
pk.yeung@ae.gatech.edu
Georgia Institute of Technology
Atlanta, GA

ABSTRACT

This paper presents new advances in GPU-driven Fourier pseudo-spectral numerical algorithms, which allow the simulation of turbulent fluid flow at problem sizes beyond the current state of the art. In contrast to several massively parallel petascale systems, the dense nodes of Summit, Sierra, and expected exascale machines can be exploited with coarser MPI decompositions which result in improved MPI all-to-all scaling. An asynchronous batching strategy, combined with the fast hardware connection between the large CPU memory and the fast GPUs allows effective use of the GPUs on problem sizes which are too large to reside in GPU memory. Communication performance is further improved by a hybrid MPI+OpenMP approach. Favorable performance is obtained up to a 18432^3 problem size on 3072 nodes of Summit, with a GPU to CPU speedup of 4.7 for a 12288^3 problem size (the largest problem size previously published in turbulence literature).

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel and high-performance simulations**; • **Applied computing** → *Physics*; • **Networks** → Network measurement.

KEYWORDS

Asynchronous, Algorithm, Turbulence, Simulations, Out-of-core, Distributed, FFT, Summit, all-to-all, Communication, GPU, CUDA, MPI

ACM Reference Format:
Kiran Ravikumar, David Appelhans, and P.K. Yeung. 2019. GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism. 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, 12 pages. https://doi.org/10.1145/3295500.3356209

1 INTRODUCTION

Many large-scale production codes running on leadership-class computing platforms have inherent needs for data movement, both within and across different parallel execution processes. Obtaining high scalability and/or satisfactory time to solution at large problem sizes is often more challenging if communication costs are dominant. In particular, in the present pre-exascale era, a key question is how codes that are communication-intensive can benefit from heterogeneous platforms whose principal advantage is fast computation on hardware accelerators such as GPUs.

Turbulent fluid flows governed by the Navier-Stokes equations with disorderly fluctuations over a wide range of scales in time and space [16] represent a major challenge in both science and computing [10–12, 22, 23]. In work focused on fundamental understanding, it is often useful to employ periodic boundary conditions on a cubic domain, with solution variables expressed in a discrete Fourier series. In pseudo-spectral methods [3] nonlinear terms are evaluated in physical space and then transformed to Fourier space, avoiding extremely costly convolution integrals. These simulations are inherently communication intensive because of the need to collect complete lines of data in the machine memory before transforms can be taken. To date, the largest simulations have reached over 1 trillion grid points, via massive CPU-based parallelism [10, 11, 23]. However, the trend toward exascale appears to favor denser nodes where future advances will likely require use of accelerator hardware and new programming approaches to optimized on-node and off-node data transfer. For example, Summit at the Oak Ridge Leadership Computing Facility (OLCF), which is currently the fastest supercomputer in the world, has fewer but much denser nodes than its predecessor machine (Titan). Utilizing GPUs instead of CPUs at very large problem sizes also presents new challenges since the amount of GPU memory is substantially less than CPU memory.

In this paper we address the nontrivial task of implementing a new pseudo-spectral turbulence code capable of reaching unprecedented problem sizes at the high throughput needed to complete long running simulations on Summit. The shift to fewer but denser nodes motivates a design strategy of hierarchical parallelism, with a fine grained parallelism mapped to GPU threads coupled to a high level parallelism managed by fewer MPI processes compared to traditional massive parallelism via CPUs.

Our overall strategy, especially at scale, is based on the expected needs to (1) improve MPI performance and to (2) use GPUs efficiently on large problem sizes. To improve MPI performance, we note that communication overhead (latency) can be reduced by using fewer MPI processes, which is well facilitated by Summit's configuration as a modest number of dense nodes. The local problem size per MPI process can be increased by utilizing the larger CPU memory without being restricted by the smaller GPU memory. We also use a hybrid MPI+OpenMP approach to further reduce the number of MPI ranks for the same problem size. In addition, we

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356209

have studied the promise of overlapping host-based communication using nonblocking MPI collectives, which provided good but not the best performance.

These considerations above render the code less communication-intensive than otherwise, and hence more readily amenable to the benefits of GPU acceleration. In order to utilize the GPUs for large problem sizes, we have implemented procedures that help hide the costs of both data movement (between the host and the device) and computation, while aggressively minimizing the costs of each. In particular, we have 1) designed an asynchronous batching strategy to overlap computation and data movement on GPU-sized pieces of data, 2) developed custom data movement kernels that are highly efficient at performing strided copies on the device, and 3) leveraged optimized NVIDIA libraries on the GPU to accelerate computations (e.g. cuFFT for 1-D FFTs).

The significant role of communication in our application implies that neither sustained flop rate nor scalability are the most relevant performance metrics. Instead, we show that the new code scales similarly as a code performing only 3D transpose all-to-all calls, with only a small overhead for actual computations. More importantly, the new code is capable of a favorable time to solution for problem sizes beyond the current state-of-the-art. The CPU to GPU speedup recorded is over 4.5X for a problem size equivalent to the largest reported in the literature to date.

In the following sections we begin with some background on equations, numerical methods and current programming approaches, while noting key differences between Summit and several existing petascale platforms. We then describe key ideas in the new algorithm in some detail, before reporting and analyzing the performance results. The new algorithm makes it possible to perform simulations of three-dimensional isotropic turbulence at a problem size of 18432^3 grid points, which (within some constraints discussed further in the sections below) is the largest problem size that is feasible on Summit for production purposes. In particular, using 3072 nodes the elapsed wall time per time step is about 14.5 seconds, which is only 50% longer than the case for a long-running 8192^3 simulation (which had about 11.4 times fewer grid points in total) using 262,144 CPU cores [23]. This new grid resolution is expected to be instrumental in further advances into fundamental understanding of turbulence, especially those which are highly dependent on the presence of a wide range of scales that are represented on a finite solution domain with higher accuracy than previously practiced in the literature.

2 EQUATIONS AND NUMERICAL METHODS

The application code in this work is written to compute fluctuating velocity fields $\mathbf{u}(\mathbf{x}, t)$ in time and three-dimensional space, according to the Navier-Stokes equations expressing the principles of mass and momentum conservation, in the form

$$\partial \mathbf{u}/\partial t + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla(p/\rho) + \nu \nabla^2 \mathbf{u} + \mathbf{f} \,, \tag{1}$$

where $\mathbf{u}$ is a solenoidal vector, p is the pressure, ρ and ν are the fluid density and viscosity (taken as constants), and $\mathbf{f}$ is a forcing term. This is a partial differential equation of the advective-diffusive type, which occurs in many studies of transport phenomena in science and engineering.

We solve Eq. 1 in a simplified solution domain which is periodic in all three directions. The velocity field is expressed in a finite-terms Fourier series, as $\mathbf{u}(\mathbf{x}, t) = \sum_{\mathbf{k}} \hat{\mathbf{u}}(\mathbf{k}, t) \exp(\iota \mathbf{k} \cdot \mathbf{x})$ where $\iota = \sqrt{-1}$, overhats denote (complex-valued) Fourier coefficients and $\mathbf{k}$ is a wavenumber vector whose coordinate components on an N^3 grid take the values $1 - N/2, 2 - N/2, \ldots 0 \ldots 1 \ldots N/2$. In Fourier (i.e., wavenumber) space Eq. 1 becomes, for a given $\mathbf{k}$, an ordinary differential equation which can be written as

$$\partial \hat{\mathbf{u}}/\partial t = -\widehat{\nabla \cdot (\mathbf{uu})}_{\perp \mathbf{k}} - \nu k^2 \hat{\mathbf{u}} + \hat{\mathbf{f}} \,. \tag{2}$$

To enforce mass conservation under the assumption of constant density, the nonlinear term is projected into a plane perpendicular to the vector $\mathbf{k}$. Aliasing errors arising from the treatment of nonlinear terms are generally controlled by a combination of phase-shifting and truncation in wavenumber space [17].

In our simulations Eq. 2 is typically integrated over many thousands of time steps, using explicit second- or fourth-order Runge Kutta (RK2, RK4) schemes for the nonlinear terms, while viscous terms are treated exactly via an integrating factor. In general RK4 offers improved accuracy and numerical stability. However, experience also shows (e.g. [17] and numerous publications in the turbulence community adopting the algorithm described therein) RK2 results are often adequate when the time step is made sufficiently small. For simplicity we have reported RK2 timings in this paper. The cost of RK4 per time step is approximately doubled, with a small increase in the memory necessary to hold partially updated values of the solution variables at different substages within a single time step.

In our code, since time advance is performed in Fourier space (per Eq. 2) each RK substep starts and ends in Fourier space, as well. The essential mathematical operations involved include transforming three velocity components to physical space, forming the nonlinear products there, and transforming those products back to Fourier space. The actual transforms are taken one direction at time. The need for data movement arises from the need to collect complete lines of data (successively, along each coordinate axis) in the core memory to be operated on.

While the number of variables being transformed varies during each time step, the structure and performance of our turbulence simulation code share many similarities with 3D FFTs, which are relevant to many science disciplines and have been the subject of much software development (e.g. [7, 9, 14]). Thus, we believe some of the novel programming techniques developed in the present work are potentially relevant to the wider computational science community, as well.

3 ALGORITHM DESCRIPTION

In this section we give a detailed description of our code development, starting with our choice of domain decomposition, and leading to our asynchronous algorithm for computing on the GPUs while using the CPU memory capacity. The communication costs of all-to-all communication, inherent in 3D FFT algorithms, dominates overall runtime especially as the FFT kernels themselves become very fast on the FLOP heavy GPUs. Because of this known communication bottleneck, which will always limit speedup no matter how many FLOPs the GPUs become capable of, algorithm design is

driven by choices that allow as much communication overlap and as large a message size as possible. We wish to emphasize that our ultimate objective is not high scalability *per se*, but instead making feasible simulations of a size exceeding the current state-of-the-art. This includes a goal of approximately 20s per RK2 timestep in order to solve long running simulations in a reasonable number of wallclock hours.

3.1 Domain decomposition

The first decision in devising a parallel algorithm for very large problem sizes is how best to distribute memory requirements among, say, P MPI processes. As seen in Fig. 1 an N^3 solution domain can be decomposed into *slabs* or *pencils*, in one and two directions, respectively. For a slab, or 1D decomposition, each MPI process works on an integer number of planes (e.g., x–z) and can take FFTs in two directions forming that plane. A global collective communication call (of the "all-to-all" type) is used to re-divide the data along the third direction (e.g. y). While the concept of a 1D decomposition is straightforward, clearly, it is limited to $P \leq N$.

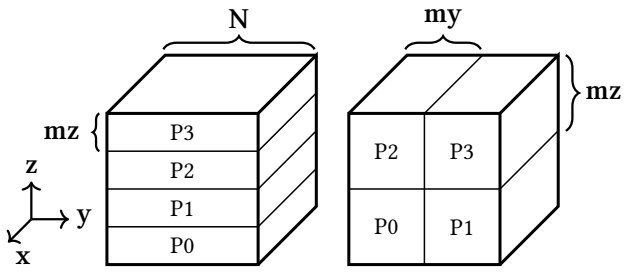

Figure 1: 1D and 2D domain decompositions, illustrated for case of MPI process count $P = 4$. On the left, each slab is of size $N \times N \times mz$, where $mz = N/P$. On the right, each pencil is of size $N \times my \times mz$, where $my = N/P_r$ and $mz = N/P_c$.

A 2D domain decomposition divides the data in two directions, allowing a finer-grained decomposition (i.e. larger P). A 2D Cartesian process grid is used, with $P = P_r \times P_c$ where P_r and P_c are the sizes of the "row" and "column" communicators, respectively. Two collective communication calls are performed using the (smaller) row and column communicators instead of once globally over P processes. The best performance is usually obtained if P_r equals the number of MPI ranks per node, since some of the communication will then occur solely on the node.

State-of-the-art turbulence simulations performed on massively parallel platforms [10, 11, 23] have generally used a 2D domain decomposition, as do the FFT portions of several of Exascale Computing Project applications (lammps, hacc) [15]. However the latest trends in HPC landscape appear to point towards machines with fewer nodes which are more powerful in both memory and speed, with Summit being a primary example. We have, accordingly, adopted the 1D (slabs) decomposition for this work, with a hybrid approach to further parallelize within a slab.

3.2 Target System and Software Stack

The target architecture around which the code was designed is the IBM Power System AC922 [21] which is used in the Summit and Sierra supercomputers. A node on Summit consists of a dual socket POWER9 processor, with each socket connected via NVLink to 3 NVIDIA V-100 GPUs (2 links/GPU) and 22 cores. Each core is capable of supporting up to 4 hardware threads. Summit nodes have 512 GB of DDR3 and each GPU has 16 GB of High Bandwidth Memory (HBM). Each V-100 GPU has 80 Streaming Multiprocessors (SMs). The overall picture is then of a very dense node with a large amount of memory and compute resources per node.

The interaction of the CPU memory bandwidth, NVLINK bandwidth, and network card bandwidth will be important in understanding some of the limitations of data movement overlap. The Power 9 CPU memory bandwidth per socket is 135 GB/s peak unidirectional, while the CPU-GPU NVLINK connection is capable of 150 GB/s (peak, per socket) and the network card on Summit is capable of 12.5 GB/s (per socket, bi-directional) [20, 21]. This means that NVLINK data transfers alone are capable of fully saturating the P9 memory bandwidth and any simultaneous use of the network card must compete with NVLINK bandwidth demands.

The code was written in Fortran and the GPUs were exercised through CUDA Fortran, implemented in the IBM XL compiler, and calls to NVIDIA's `cuFFT` library. CUDA streams and CUDA events were used to control the asynchronous tasks of batching data on/off the GPU, computing FFTs, and determining when the data was available in host memory to be sent through asynchronous `MPI_IALLTOALL` calls.

3.3 A basic (synchronous) GPU algorithm

Prior to a detailed description of our best-performing asynchronous algorithm, it is useful to review the basic requirements for a GPU implementation, in terms of the work needed to perform a complete 3D FFT. Figure 2 shows the basic sequence of operations, focused on how a 3D FFT is taken of solution variables initially in Fourier space. This sequence is also a close match with the work performed in the first half of each Runge-Kutta substage in our turbulence code, before the nonlinear terms are formed and transformed back to Fourier space. The sequence of operations shown in Fig. 2 also reverts to that of a CPU code if the host-to-device (H2D) and device-to-host (D2H) data copies are eliminated and all operations are carried out on the CPUs.

At the beginning of the execution sequence in Fig. 2, each MPI process holds a slab of data that consists of x–y planes stacked up in N/P units in the z direction. This slab of data is copied from host to device. Next the 1D FFTs in the y direction are computed on the GPU using `cuFFT`. An all-to-all communication is then required to transpose these partially-transformed quantities into slabs of $x - z$ planes. Since the data to be exchanged is not contiguous in the local memory, we have to either (a) pack the data into contiguous messages locally, or (b) use MPI derived datatypes. We compared the performance of packing on the CPU, packing/moving through the use of GPU zero copy kernels, and packing on the GPU and then copying to the CPU. The fastest results were obtained by performing the packing on the GPU and then copying from device to host, and then having the host perform an `MPI_ALLTOALL`. Subsequently

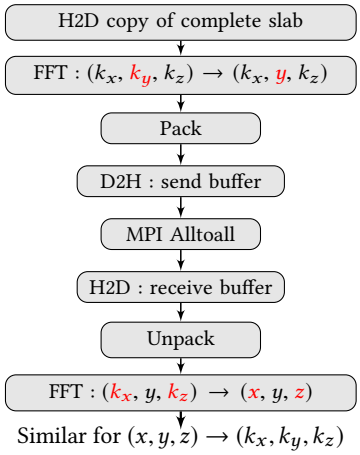

Figure 2: Schematic showing the different operations involved in computing 3D FFTs using GPUs in a synchronous manner. Similar operations in the reverse order are required to transform back to Fourier space.

the transposed data is copied from host to device, unpacked into the correct memory locations, and transformed in z and then x. Recent advancements in software and hardware, such as CUDA-aware MPI and GPU-Direct [19], allow MPI to directly transfer data that is resident on the GPU. In theory such a strategy should help avoid the additional data transfers before and after the global transpose. However, since without GPU-direct we are already achieving transfer rates at close to peak system memory bandwidth and are limited most by the bandwidth of the network interface card, the expected benefits of GPU-direct may be insignificant. In practice, after implementing CUDA-aware MPI and GPU-direct we did not see any noticeable benefit to our runtime.

Since the Fourier coefficients (being of real-valued variables) satisfy the property of conjugate symmetry, i.e., $\hat{\mathbf{u}}(-\mathbf{k}) = \hat{\mathbf{u}}^*(\mathbf{k})$, the transforms are complex-to-complex in two directions (y and z) but complex-to-real in the third (x). This ordering of the transform directions (y,z,x; reversed when transforming from physical to wavenumber space) is chosen so that formation of nonlinear terms in physical space can be performed more efficiently on arrays of stride unity. In our data structure, FFTs in x are more efficient because of a unit vector stride. For FFTs in y and z we have the options of performing these transforms in a strided or unstrided data structure, but on Summit we have found that either option takes about the same time when the cost of additional local data reordering is also factored in.

The algorithm illustrated here is readily extended to the pseudo-spectral turbulence code, in which a reverse sequence of operations takes place after the nonlinear terms are formed. However, there are two hurdles to direct application of the above algorithm. The first is that the algorithm is synchronous, such that each set of operations is carried out sequentially without taking advantage of the fact that operations on different portions of data can be made to occur asynchronously (e.g. [5]). The second is that, as written, each GPU is to process a single slab of data all at once, which creates difficulties in processing larger problem sizes where a single slab of data will not fit into the GPU memory. To compute at larger problem sizes (as is our goal) it is beneficial to break a slab into smaller portions that will fit into the GPU memory. This data division inside a slab opens up the possibility of a significant degree of task asynchronism, where, for instance, different planes (or even parts of planes) within a slab may be copied, computed, and communicated simultaneously. Indeed, this is the motivation in the development of an asynchronous algorithm capable of larger problem sizes, as described in the next subsection.

3.4 Batched asynchronous algorithm

Our objective is to be able to run large problem sizes efficiently without being limited by the GPU memory capacity. The GPU memory issue can be addressed by dividing a slab into several (np) pencils and processing each pencil separately, as illustrated in Fig. 3. This also provides an opportunity for overlapping operations on different pencils within the same slab.

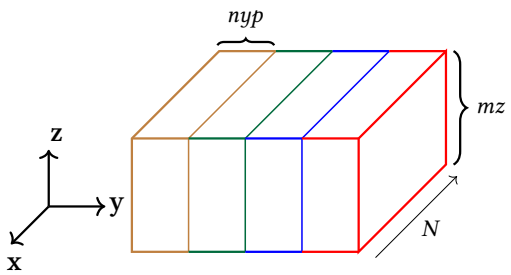

Figure 3: Decomposition of a slab of data into multiple (np) pencils, each of size $N \times nyp \times mz$, where $nyp = N/np$, to enable larger problem sizes where a single slab does not fit into the GPU memory.

To enable the desired asynchronism we use the programming model of CUDA streams and events. We specify two separate *CUDA streams*, one for computations and one for data movement. Besides allowing overlap with data movement and compute, a distinct data transfer stream ensures that bandwidth is devoted to one direction of traffic at a time. This is beneficial on Summit, because while NVLink supports both maximal read and maximal write bandwidth simultaneously, the host memory bandwidth supports a combined read *or* write bandwidth and the maximum aggregate bandwidth is achieved when performing unidirectional movement [20]. Our choice of a single transfer stream allows us to devote the full bandwidth to whichever transfer operation is first put into the stream, ensuring that the right piece of data is copied into the GPU as quickly as possible. *CUDA Events* are used to enforce synchronization between operations in different streams [18].

Figure 4 shows the sequence of operations in the new asynchronous algorithm. For brevity we are showing only the operations needed to transform from Fourier space to physical space (with those from physical to Fourier space being very similar but reversed in order). Color coding is used to help identify operations in the transfer stream (H2D and D2H copies), compute stream (such as FFTs in separate directions), and communication. Colons within a

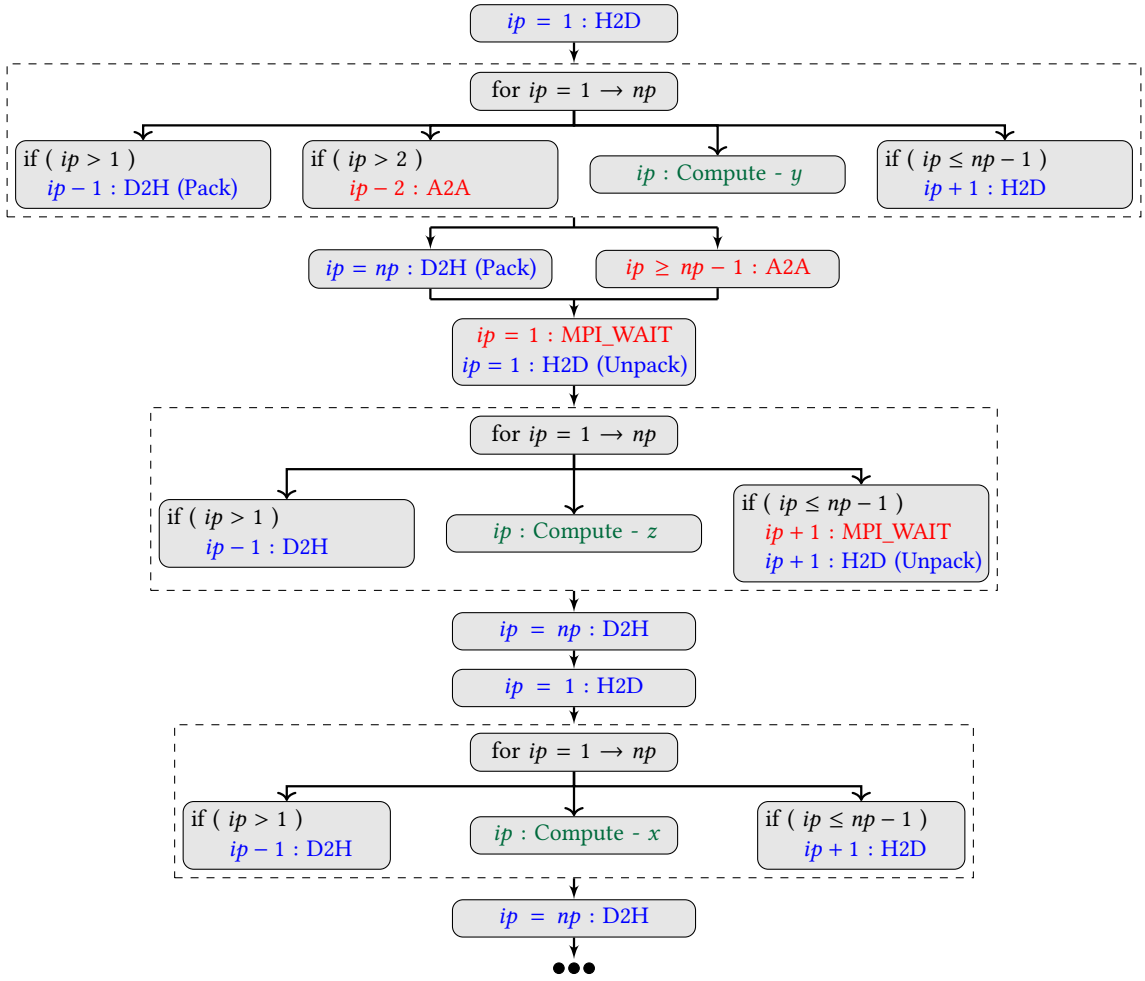

Figure 4: Schematic showing the asynchronous GPU algorithm where operations on the same row are performed asynchronously. The operations in blue are performed in the transfer stream while the operations in green are in the compute stream and the operations in red are using the network. The compute operations correspond to Fourier transforms and other computations such as forming non-linear products in the DNS code.

box refer to operations carried out on a specific pencil whose ID, denoted by ip, ranges from 1 to np. The three large regions bounded by dashed lines correspond to computations carried out in the y, z and x directions, respectively. Data transfer operations on the first and last two pencils are handled separately as shown by the blocks outside the large dashed regions.

Before entering the first dashed region the first pencil is copied to the GPU. Then, if $ip = 1$ the code performs a compute. When $ip > 1$ the $(ip-1)^{th}$ pencil is copied back from the GPU and packed into a contiguous array on the CPU, provided the computations on it have been completed. The pack operation is performed as a strided data copy to avoid packing the data before the global transpose. This way both the packing and the D2H are performed in a single operation. Although operations on the same row are executed asynchronously, our operations are launched from left to right, which is to prioritize data copy out of the GPU so that the global transpose can be initiated as soon as possible. A non-blocking MPI all-to-all is launched on the $(ip-2)^{th}$ pencil only when the D2H copy of the $(ip-2)^{th}$ pencil is completed. Computations are performed as soon as the H2D copy on the ip^{th} pencil is completed. A H2D copy for the next pencil is also posted at this time. If ip equals one of its last two values the code takes special action to copy out the last pencils back to the CPU, and to post the global transposes.

It should be noted that copy operations in the transfer stream are performed asynchronously, i.e., the CPU can move forward to other tasks but it does not imply the copy has completed or even started. An *event* is recorded to track the progress of the copy. This ensures the D2H copy will begin only once the computations on the $(ip-1)^{th}$ pencil are completed. Similarly, the H2D copy waits for the data in the GPU buffer into which the host data needs to be placed is copied out.

Operations in the second and third dashed regions, performed on data in $x-z$ slabs, are also scheduled in a manner that allows

overlapping between the transfer and compute streams. The only MPI function call from here until completion of the entire 3D FFT sequence is an `MPI_WAIT` in the second dashed region. This is to ensure the transpose completes before the H2D copy is posted.

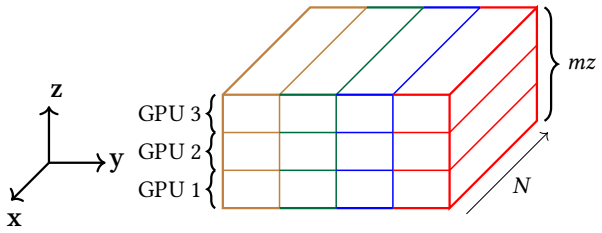

Figure 5: When running with multiple GPUs per MPI rank, each pencil is further divided up vertically to allow running with multiple GPUs per MPI task. The pencil fractions are processed by the different GPUs available to the MPI rank.

As to be noted in Secs. 4 and 5, there is some advantage in using OpenMP threads instead of pure MPI on the CPU. On Summit, when the number of MPI ranks per node drops below 6, multiple GPUs per MPI rank will become available. Each pencil is further divided vertically such that a fraction of the pencil is run on each GPU as shown in Fig. 5. One OpenMP thread per GPU is used to launch the same operations as described in Fig. 4 to the different GPUs available to each MPI rank. The device each thread works with is set using the `cudaSetDevice` API call. The global MPI transpose is posted only after the entire pencil has been processed by each of the GPUs available to the MPI rank. The code is also capable of waiting for all the pencils in a slab to be processed so that one large all-to-all can be posted instead of multiple smaller ones. The logic of the algorithm in Fig. 4 is still applicable.

3.5 Problem sizes and node counts

It is useful to develop estimates of the node count necessary to meet the memory requirements of a chosen problem size. Our focus is on solving the largest problem possible on Summit, subject to some constraints. The first constraint is that we prefer a wallclock speed on the order of 20 seconds per RK2 timestep because this allows a reasonable simulation turnaround time in human hours. The second constraint is that N be powers of 2 or at least an integer rich in factors of 2 because this usually leads to the best discrete FFT library performance. Furthermore N should be evenly divisible by 3 to facilitate even division among Summit's 3 GPUs per socket. We choose $N = 18432$ as our target because it is rich in factors of 2, divisible by 3, and (as will be shown below) it will fit in Summit's memory.

For an N^3 problem involving D variables at single precision on M nodes, the memory required per node is $4DN^3/M$ bytes. A detailed counting of the number of velocity components, nonlinear terms, and send/receive buffers which are used to transfer data between CPUs and GPUs, yields $D \approx 25$. These buffers are page-locked and cannot be swapped to disk, because they are allocated as *pinned* memory. From experiments, we estimate that the operating system occupies approximately 64 GB on each Summit node, leaving 448 GB for user codes. Equating $4DN^3/M$ (where $D = 25$ and $N = 18432$) to 448 GB gives $M = 1302$, which is the minimum number of nodes needed. However, for load balancing on a per-node basis the number of nodes should be a factor of N. With $N = 18432$ and noting the total system on Summit has approximately 4608 nodes, the only 2 possible values of M are thus 1536 and 3072. In the interest of a shorter time-to-solution we use 3072 nodes which is 67% of the full system.

For a given CPU node count we also need to consider how the memory might fit into the GPUs, generally by processing pencil-sized portions of each slab at a time. For compute purposes, 9 pencil-sized buffers are required. This number needs to be tripled, to 27, to allow asynchronous execution in the manner described in Sec. 3.4, while pack and unpack operations can be performed without additional buffers. If we have np pencils per slab then each pencil contains $N^3/(M \times np)$ words (per variable). As for the GPU memory we assume 96 GB of GPU memory on each node is user-accessible, with no systems-related tasks running on the GPU. Thus, equating $4 \times 27 \times N^3/(M \times np)$ bytes (with $N = 18432$, $M = 3072$) to 96 GB gives, nominally, $np = 2.13$. In practice, because further needs for memory also arise from other smaller arrays, for $N = 18432$ we find that np needs to exceed 3.

Since np must be an integer, we conclude that each slab in the 18432^3 problem has to be divided up into a minimum of 4 pencils to fit in the GPU memory. The node count and the number of pencils required for a range of problem sizes are given in table 1.

Table 1: Node counts, problem sizes, minimum number of pencils per slab and the size of each pencil (for 1 variable) that can fit into the GPU memory. Problem sizes from 3072^3 to 12288^3 are exact weak scaling cases, while 18432^3 is larger than what weak scaling suggests.

# Nodes	Problem size	Mem. occ. per node (GB)	No. of pencils	Size of pencil (GB)
16	3072^3	202.5	3	2.25
128	6144^3	202.5	3	2.25
1024	12288^3	202.5	3	2.25
3072	18432^3	227.8	4	1.90

4 CODE OPTIMIZATIONS

In our code development effort we have focused especially on cross-node data communication and also on-node data movement between the CPU (host) and GPU (device). In the two subsections below we discuss several different approaches and the performance data obtained for them on Summit.

4.1 MPI Configurations

Given Summit's node architecture, two natural choices for the number of MPI tasks per node are 6 (one per GPU) or 2 (one per socket). In the latter case, OpenMP threads can be used to launch operations to the 3 GPUs per socket, while the message size per MPI rank is increased by 3X. In addition, in both scenarios above, since the slab of data assigned to each MPI rank is further broken down into pencils of data which are batched on and off of the GPU for processing, each MPI rank can be made to communicate the

entire slab all at once, one pencil at a time, or a selected number (say, Q) of pencils per call. The choice $Q = 1$, where an MPI all-to-all is called to transpose a pencil of data as soon as the pencil has been processed and copied back from device to host, is conducive to overlapping MPI with data movement and computation. However it also leads to messages that are potentially so small that they become dominated by latency. Fewer MPI calls with larger messages can be realized by choosing an non-unity value of Q, up to the number of pencils present in each slab.

The network interconnect on Summit is a dual-rail EDR InfiniBand network and provides a node injection bandwidth of 23 GB/s [8] and a bisection bandwidth of 46GB/s. Although these bandwidths are in principle achievable for point-to-point communications of sufficient size, for all-to-all communication the bandwidth achieved at scale can be considerably lower, because the individual peer to peer (P2P) messages involved can become very small. In our code, if each slab is divided into np pencils, then the message chunk that must be delivered to each process (P2P message size) for nv variables at single precision is $4 \times nv \times (N/np) \times (N/P)^2$ bytes.

To understand the MPI performance we have conducted tests using a standalone MPI all-to-all kernel which carries out communication operations mimicking those in the DNS code but do not compute nor move data between CPU and GPU. One key difference is that whereas the DNS code uses non-blocking `MPI_IALLTOALL` to allow for overlapping between MPI and local operations, the standalone kernel instead calls the blocking `MPI_ALLTOALL`, which is important for clean MPI performance data to be obtained. A collection of MPI performance data is shown in Table 2, where problem sizes and node counts correspond to the information in Sec. 3.5. The effective bandwidth is calculated by the formula

$$BW = (2 \times P2P \times P \times tpn) \; / \; time \tag{3}$$

where tpn is the number of MPI ranks per node, and a factor of 2 is included since all-to-all's are comprised of both sends and receives. This formula includes on-node messages in the computation of the bandwidth, but this simplification becomes insignificant at larger problem sizes.

Table 2: Effective bandwidth per node of MPI all-to-all on Summit at different node counts. The message size communicated between each MPI process (P2P) is reported (for 3 variables).

Nodes	A: 6 tasks/node		B: 2 tasks/node		C: 2 tasks/node	
	1 pencil/A2A		1 pencil/A2A		1 slab/A2A	
	P2P (MB)	BW (GB/s)	P2P (MB)	BW (GB/s)	P2P (MB)	BW (GB/s)
16	12	36.5	108	43.1	324	43.6
128	1.5	24.0	13.5	39.0	40.5	39.0
1024	0.19	11.1	1.69	23.5	5.06	25.0
3072	0.053	13.2	0.47	12.4	1.90	17.6

We refer to the three cases in Table 2 as A, B, C, respectively. Between A and B, the P2P message size increases by 9X because data per process is tripled and there are also 3 times fewer processes to perform the data exchange. The increase in P2P message size from B to C is a factor of np since there are np pencils per slab. Case B gives a higher bandwidth achieved than case A, up to a node count of 1024. At 3072 nodes it is surprising that case A performs slightly better than case B, because this departs from 1) the trend of larger P2P message size leading to better bandwidth and 2) our experience with the full DNS code where case B has a faster solution time than case A. However, as noted above, the DNS code uses non-blocking MPI all-to-all calls, whereas the standalone MPI code uses the blocking version. One possibility is that at sufficiently small message sizes and without overlap, case A may be able to take advantage of eager limits and hardware acceleration in the network.

The trend in increased P2P message size leading to increased bandwidth continues when comparing B to C, especially at scale. This is expected because, in general, a shift from a larger number of smaller messages to a smaller number of larger messages reduces the effect of network latency. All three cases were implemented in the turbulence code, and as was observed in the standalone MPI tests, the configuration of case C led to the best-performing implementation of the turbulence code as well, which is expected because of the communication-intensive nature of these codes.

4.2 Strided copy optimization

Our batched asynchronous algorithm described in Sec. 3.4 requires frequent strided copies of relatively small units of data between host and device. This pattern can be expected for any algorithm performing line operations on a distributed 3D domain. Such strided copies occur when data stored on the CPU as either $x - y$ or $x - z$ planes must be moved onto the GPU in smaller pieces, or when being packed in preparation for MPI all-to-all communication.

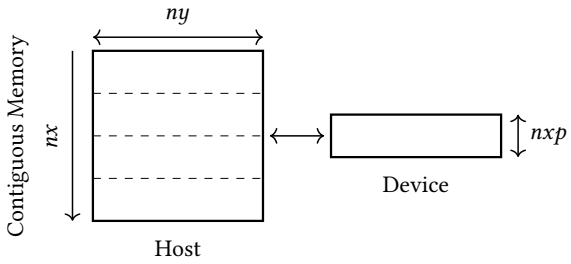

Figure 6: Top down view of a slab of data on the CPU and a pencil of data on the GPU where $nxp = nx/np$. Strided (in y) copies of contiguous data (in x) is required in order to transform in y direction since memory is linear (stride 1) in the x direction on both the CPU and GPU. Strided FFTs are performed in the y direction to avoid reordering on the GPU.

For example, for a $x - y$ slab divided into 4 pencils as shown in Fig. 6, copying a pencil of data to the GPU requires copying a series of contiguous chunks of data to the GPU. For the 18432^3 problem the innermost dimension to be copied will have $18432/4 = 4608$ elements or 18 KB of contiguous memory. An entire pencil of such lines of data must be copied, i.e., since $mz = 3$ and $nv = 3$ the pencil shape is $4608 \times 18432 \times 3 \times 3$. This gives 165888 chunks of 18 KB each, which must be copied. When the chunk size is small (and the number of chunks therefore large), the many `cudaMemCpyAsync`

calls required can be very slow, presumably because the API call overhead begins to become significant. A high performing solution to this is potentially using a CUDA zero-copy kernel to have the GPU instead of the CPU initiate the many small transfers on host page-locked memory [2]. Therefore, for this work, we tested two alternative approaches, namely custom written "zero-copy" CUDA kernels and an asynchronous version of the CUDA library call `cudaMemCpy2D`, as described below.

The custom zero-copy CUDA kernel uses threads to move data between arrays allocated in the GPU memory and arrays which reside in host memory. The zero-copy kernel makes use of the fact that CUDA threads can access host resident memory directly from the GPU without having to explicitly copy the data, i.e., there are zero copies living on the device. This is enabled by using the CUDA library call `cudaHostGetDevicePointer` to acquire a device valid pointer to pinned host memory. This host memory must be page-locked memory (pinned) which is also needed for maximum transfer speeds for host arrays that are frequently copied in and out of the GPU.

The zero-copy kernel method, however, is limited by the fact that it uses some of the GPU streaming multiprocessors to copy data, which can slow down the other computational kernels. On the other hand, the CUDA library call, `cudaMemCpy2DAsync`, uses the GPU copy engines and accepts arguments that allows for simple strided copies to be performed easily. This comes with the advantage of not having to occupy GPU SMs to achieve the movement.

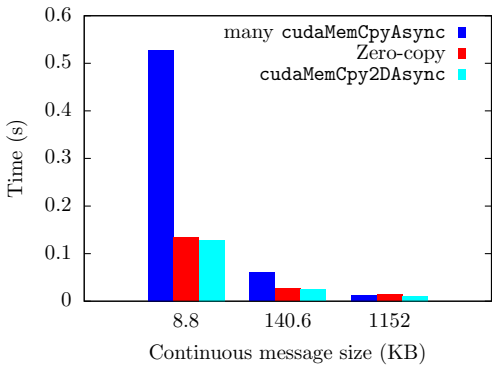

Figure 7: Time to transfer a total of 216MB of data with strided memory access using three different approaches. Since the total pencil size is fixed, smaller contiguous messages in this plot also correspond to more required looping over contiguous message chunks.

Figure 7 compares timings for strided memory copies obtained from the three different approaches considered here. The copies were performed on a fixed total message size of 216 MB but the size of the contiguous memory in the strided copy is varied. It can be seen that both the zero-copy and `cudaMemCpy2DAsync` approaches perform much better than (many) `cudaMemCpyAsync` when the contiguous message sizes are below 100's of KB. For the 18432^3 DNS problem the contiguous extent of the pencils is 18 KB, which is close to the 8.8 KB tested in the figure. From this data we can draw two conclusions. The first is that the many `cudaMemCpyAsync` approach is much slower than the zero-copy or `cudaMemCpy2DAsync` approaches, while the latter two gives similar timings. The second is that when moving a fixed amount of data, the overhead involved in moving a finer granularity of chunks can increase the movement time.

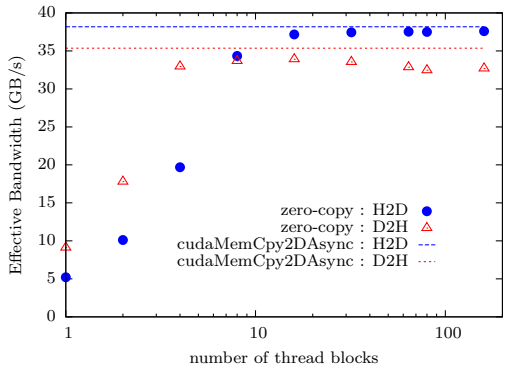

Figure 8: Effective bandwidth of zero-copy kernel (circles or triangles) compared to using `cudaMemCpy2DAsync` (dashed horizontal lines) for different numbers of thread blocks. The size of the thread block was 1024 threads.

To understand the GPU resources required by the zero-copy kernel to provide sufficient throughput we studied the behavior of the zero-copy kernel using different numbers of CUDA blocks as shown in Fig. 8. The kernel register usage of 32 registers per thread and the chosen thread block layout of 128×8 threads per block, allows two blocks to occupy a single SM. The `cudaMemCpy2DAsync` is a CUDA API call, not a CUDA kernel, and therefore does not use any blocks (or SMs). If the zero-copy kernel is provided with sufficient GPU resources, the bandwidth achieved is similar to `cudaMemCpy2DAsync`. We also see that close to maximum throughput is attained even if using only a small fraction (about 16 blocks) of the GPU resources. This allows a small fraction of the GPU resources to be devoted to a zero copy kernel while simultaneously running other compute kernels on SMs of the GPU.

For best overall performance, `cudaMemCpy2DAsync` is still preferable to zero-copy kernels since the former allows all the GPU resources (streaming multiprocessors) to be available to the compute kernels. However, `cudaMemCpy2DAsync` can only handle simple strides, while the zero-copy kernel can handle data with complex stride patterns, (e.g., in unpacking data from contiguous to non-contiguous arrays after communication), while using up only a small amount of GPU resources. Thus, in our production code, most of the copying between host and device is implemented using `cudaMemCpy2DAsync`, while data unpacking is performed using the zero-copy kernel.

5 PERFORMANCE ANALYSIS

In this section we present and analyze the overall performance of the newly developed DNS code at different problem sizes and node counts as described in Sec. 3.5, with all data obtained on

Table 3: Performance of the slab decomposed DNS code run under different configurations and speedups are calculated with respect to the performance of the pencil decomposed synchronous CPU code.

Nodes	Problem Size	Sync CPU	Async GPU					
			6 tasks/node		2 tasks/node			
					1 pencil/A2A		1 slab/A2A	
		Time (s)	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
16	3072^3	34.38	8.09	4.2	6.70	5.1	7.50	4.6
128	6144^3	40.18	12.17	3.3	8.66	4.6	8.07	5.0
1024	12288^3	47.57	13.63	3.5	12.62	3.8	10.14	4.7
3072	18432^3	41.96	25.44	1.6	22.30	1.9	14.24	2.9

Summit. Table 3 shows elapsed wall time per time step and speedup relative to performance data collected using the synchronous pencil decomposition CPU code that was used by the authors of [23] and is based on the principles described in Sec. 3.2. Timings per step were obtained by taking the maximum over all MPI ranks, averaged over multiple time steps. It should be noted that for both CPU and GPU codes, regardless of the domain decomposition chosen, load balancing requires that the number of cores used per node should be an integer factor of the linear problem size (N). This implies, even though there are 42 cores per Summit node, only 32 cores can be used for most problem sizes except for the 18432^3 problem which allows 36 cores when run with 3072 nodes.

The best MPI configuration in our new asynchronous GPU algorithm gives the time to solution at a resolution of 18432^3 grid points, using 3072 nodes, at under 15 seconds per time step. To our knowledge, considering the problem sizes involved this compares favorably with the largest simulations performed in the recent past [10, 23] using CPU-based massive-parallelism. A GPU-to-CPU speedup close to 3X was observed for the 18432^3 problem. Given the large problem size addressed and the communication-intensive nature of our application, this speedup is substantial.

5.1 2 vs 6 tasks per Node

The DNS code was approximately weak scaled on Summit, starting with a problem size of 3072^3 on 16 nodes up to 18432^3 on 3072 nodes. The scaling is approximate because the number of MPI ranks must be an integer factor of the number of grid points on each side of the domain, while we are focused on simulating the largest problem size that can fit into the memory available on the machine, at scale. Timings are reported for the three MPI configurations defined earlier in Table 2. In particular: (A) using 6 tasks per node and communicating one pencil at a time; (B) using 2 tasks per node and communicating 1 pencil at a time (overlapping MPI with GPU movement and compute); or (C) using 2 tasks per node but waiting to communicate until all pencils in the slab have been computed and can be sent in a larger message (no MPI overlap with GPU operations). In all three cases data movement to/from the GPU and computation on the GPU are overlapped with each other, in the manner described in Sec. 3.4. As might be expected for a communication dominated code, performance comparisons between these three MPI configurations follow the trends observed in the achieved MPI bandwidth studies of Table 2. Namely, that 2 tasks per node perform better than 6 tasks per node and that at 3072 nodes sending an entire slab per all-to-all performs better than asynchronously sending each pencil as they become ready. Figure 9 shows the GPU timing data as well as a standalone MPI code which only performs the transposes without computation or data movement between the host and device. This helps us estimate standalone MPI costs; for example the difference between the red line and dotted green line of figure 9 is time spent in non-MPI activities, such as GPU kernels and GPU data transfer. This green line provides an upper bound on the best possible performance given the network characteristics of the machine we used. Faster GPUs or optimization to the GPU kernels alone can at best approach the performance of the dotted green line.

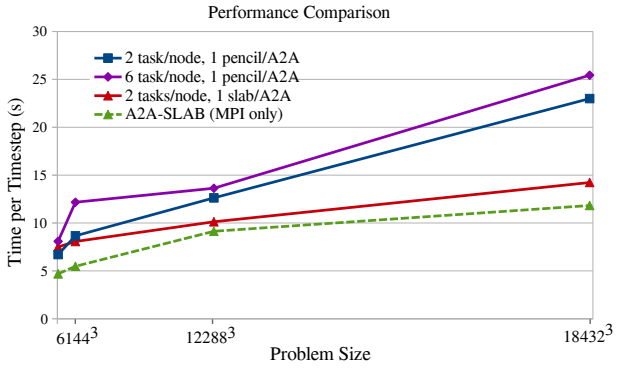

Figure 9: Time-per-step of the DNS slab code. The dotted green line is a benchmark of performing only the required MPI all-to-all calls (no computations). The solid lines are runs of the DNS code in various configurations benchmarked up to 3072 Summit nodes.

5.2 Timeline and Asynchronous MPI Analysis

Use of The NVIDIA visual profiler [13] coupled to a Fortran interface to Nvidia's nvtx markers [1] allows the visualization of a timeline of asynchronous CPU and GPU operations. To better understand the performance differences of the code under the various configurations, Fig. 10 shows plots of normalized and aligned timelines for the various configurations running on 1024 nodes. These timeline plots are particularly illuminating because they directly show the parts of the code which contribute to the performance differences.

For example, the MPI time (shown in red) is immediately seen to be the major user of runtime. Comparison between the MPI times of

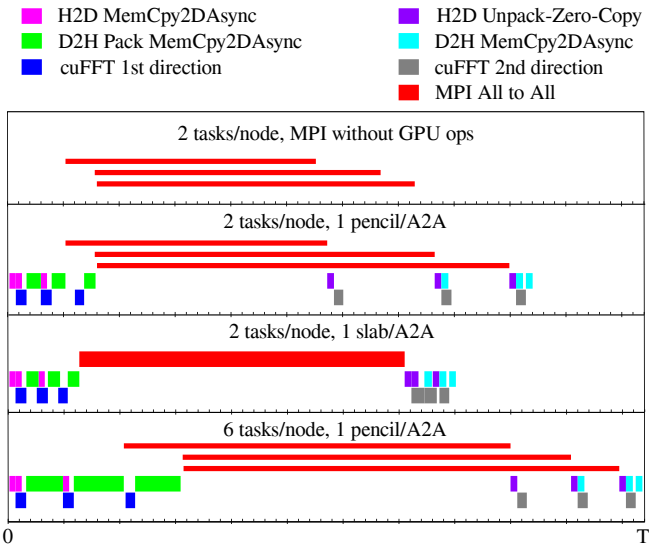

Figure 10: A normalized timeline comparison of various code configurations running the 12288^3 problem size on 1024 nodes. Each slab is divided into 3 pencils. The top timeline shows the behavior of an MPI only code that communicates the 3 pencils at the same points in time as the DNS code in the second timeline. The second timeline shows the actual behavior of the DNS code where GPU operations are running asynchronously with the all-to-all communication. The third timeline shows the affect of waiting to send the entire slab in one MPI all-to-all call instead of each pencil as it becomes ready. The final timeline shows the behavior of 6 tasks per node when each pencil is asynchronously sent as they become ready.

the two uppermost timelines shows that MPI in the actual DNS code takes somewhat longer than in the standalone MPI code. Reasons for this are not fully understood, but the results were very repeatable. A known contributor is that both CPU-GPU data movement and MPI data movement share the total bandwidth limit of the Power 9 memory. Our standalone tests revealed that if GPUs and the network card were requesting data movement, the MPI bandwidth suffered significantly until the GPU transfer was complete. However, even if the GPU data transfer times are subtracted from the MPI time in the second timeline, the MPI time still does not equal the MPI time without GPU operations. The difference in timing is not fully understood, but another important consideration is that these timelines were generated using NVIDIA's profiler which has non-trivial overhead and file system resource demands.

Comparison between the second and third timelines shows that the same amount of data can be transposed faster when processed as one, larger, message. This highlights a trade off: GPU operations can be overlapped with MPI communication of individual pencils (second timeline), but the resulting MPI message sizes will be smaller and must compete with GPU data movement bandwidth demands. Both factors work to slow down the MPI operations. The best approach is dependent on the problem size. At large problem sizes the individual message sizes that comprise the all-to-all become very small and the effective bandwidth drops (see Table 2). Beyond 16 nodes, waiting to send the entire slab at once (1 slab/A2A) is faster than overlapping computation with communications of a pencil at a time (1 pencil/A2A).

In the (bottom) timeline for the 6 tasks per node case each MPI call takes longer than those seen in the 2 tasks per node case (top). This is because the P2P message size in this all-to-all is small and there are more MPI tasks with which to communicate with, which increases latency costs. An additional drawback for this 6 tasks per node case is that the D2H packing *MemCpy2DAsync* section of code takes much longer. This is because for the pack operation, the number of times cudaMemCpy2DAsync must be called is directly proportional to the number of tasks. Per GPU, the 6 tasks per node and 2 task per node cases pack the same total size buffer, but with 6 tasks per node the packing must be done at a finer granularity. The number of copies required is now 3X that for the 2 tasks per node case. This results in increased overhead as seen in Sec. 4.2. A zero-copy kernel can be used here but it degrades the performance of the 2 tasks per node case by stealing GPU resources from the compute kernels.

The last takeaway from these timelines is that for the 2 task per node cases, the MPI cost is dominating the runtime of the code. The overhead incurred in choosing to batch data between CPU and GPU is not significant compared to the total runtime, yet by batching we are able to solve using the much larger CPU memory. Further gains in performance will depend on code redesigns and hardware innovations that improve the performance of the all-to-all communication.

5.3 Scalability

As noted throughout this paper, our core objective is to solve large problems in a reasonable amount of time. In most cases, for each node count we have attempted to solve the largest problem that will fit into the memory. As a result, at each problem size we can collect timings over only a narrow range of node counts, making inferences on strong scaling of limited relevance in this work. We therefore focus on a brief discussion of weak scaling here.

The significant communication costs of codes that depend on large scale 3D FFTs imply perfect weak scaling is not achievable [4, 6, 7, 14]. After speeding up the FFT and auxiliary kernels through use of an accelerator, our code runtime is dominated by all-to-all communication. MPI benchmarks given in Table 2 indicate that algorithmic choices that leads to a small number of large messages are usually beneficial. However, eventually at the large scales the latency of small message sizes gives rise to longer MPI communication times (and decreased bandwidth).

Table 4: Weak scaling relative to 3072^3 problem size.

Nodes	Ntasks	Problem Size	# pencils per A2A	Time (s)	Weak Scaling (%)
16	32	3072^3	1	6.70	-
128	256	6144^3	3	8.07	83.0
1024	2048	12288^3	3	10.14	66.1
3072	6144	18432^3	4	14.24	52.9

We calculate the weak scaling percentage (WS), based on problem size, between two problem sizes N_1^3, N_2^3 of node counts M_1, M_2 with execution times t_1 and t_2 respectively using the formula

$$WS = \frac{N_2^3}{N_1^3} \times \frac{t_1}{t_2} \times \frac{M_1}{M_2} \ . \tag{4}$$

Table 4 shows weak scaling computed with respect to the 3072^3 (16 node) problem size, using the best performance timings for each problem size as recorded earlier in Table 3. Considering that between 3072^3 and 18432^3 the number of grid points has increased by a factor of $6^3 = 216$, we argue that a weak scaling around 53% is very respectable, given that it is a pseudo-spectral code dominated by all-to-all communication patterns.

In passing, we also examined the strong scaling of the 6 tasks/node configuration. For the 18432^3 problem, using 3072 nodes achieved 25.4s per timestep, while 1536 nodes achieved 48.7s, resulting a strong scaling of 95.7%.

6 CONCLUSIONS

In this paper we have reported in detail on design and performance aspects of a new GPU algorithm for direct numerical simulations (DNS) of turbulent flow, optimized for the dense node architecture of Summit, a 200 Petaflops pre-exascale computer which is currently the world's fastest. The best implementation of this algorithm gives a favorable time to solution for a problem size of 18432^3 grid points, of under 15 seconds of wall clock for each second-order Runge-Kutta time step. This resolution is higher than that reported in current state-of-the-art simulations, which mostly employed CPU-based massive parallelism. Speedup measured relative to the best-performing CPU code is of order 3 or higher for all problem sizes tested.

Using the latest GPU data movement techniques allows efficient use of the full node memory, which in turn allowed for solving larger problems and larger MPI message sizes. A close examination of code region runtimes (Fig. 10) shows that, as a result of powerful GPUs and fast NVLink connections, the cost of FFT computation and data movement between CPU and GPU is reduced to less than one-seventh of the code runtime. The bulk of the remaining runtime is spent on network all-to-all communications, which was also studied independently using a standalone code (Sec. 4.1).

A one-dimensional decomposition combined with a hierarchy of MPI+OpenMP parallelism is used to allow communication in the form of a smaller number of larger messages, which is crucial for achieving acceptable scaling performance, especially at larger problem sizes. The new code features capability to asynchronously overlap compute, GPU-CPU data movement, and MPI communications (Fig. 4). However, at node counts of 128 and greater, performing MPI asynchronously become more expensive than simply waiting for the entire slab of data to be processed before initiating the MPI all-to-all.

The lessons learned as well as successes achieved in this work are directly relevant to large computations in many science domains where 3D Fast Fourier Transforms are useful, and in fact generalizable to a variety of large production use codes characterized by substantial needs in communication. We have shown it is possible to efficiently utilize the very large CPU memory while still extracting substantial benefits from the GPU as an accelerator. Continuing research in achieving higher communication performance on leadership computing platforms is still vital in the pre-exascale era and beyond.

ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a Department of Energy (DOE) Office of Science user facility supported under Contract DE-AC05-00OR22725. We gratefully acknowledge use of advanced computing resources at the OLCF under a Summit Early Science Award and an INCITE 2019 Award. The DOE-IBM-NVIDIA Center of Excellence at Oak Ridge was instrumental in this work. We also appreciate much encouragement and valuable technical input from many OLCF staff members, including R. Budiardja, O. Hernandez, J.C. Hill, J. Larkin, and J.C. Wells. In addition, the first author is supported by Oak Ridge National Laboratory. (Monitor: O. Hernandez).

REFERENCES

[1] D. Appelhans. 2018. Github repository: gpu-tips/nvtx/nvtx_mod.F90. (2018). Retrieved April 10, 2019 from https://github.com/dappelha/gpu-tips/tree/master/nvtx

[2] D. Appelhans. 2018. Tricks, Tips, and Timings: The Data Movement Strategies You Need to Know. In *GPU Technology Conference*. http://on-demand.gputechconf.com/gtc/2018/presentation/s8557-tricks-tips-timings-the-data-movement.pdf

[3] C Canuto, M.Y. Hussaini, A. Quarteroni, and Zang T.A. 1988. *Spectral Methods in Fluid Dynamics*. Springer-Verlag, Berlin.

[4] A. G. Chatterjee, M. K. Verma, A. Kumar, R. Samtaney, B.. Hadri, and R. Khurram. 2018. Scaling of a Fast Fourier Transform and a pseudo-spectral fluid solver up to 196608 cores. *J. Parallel and Distrib. Comput.* 113 (2018), 77–91.

[5] M. P. Clay, D. Buaria, P. K. Yeung, and T. Gotoh. 2018. GPU acceleration of a petascale application for turbulent mixing at high Schmidt number using OpenMP 4.5. *Comp. Phys. Comm.* 228 (July 2018), 100–114.

[6] H. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P. K. Yeung, and R. Vuduc. 2012. On the communication complexity of 3D FFTs and its implications for exascale. In *Proceedings of the International Supercomputing Conference (ISC'12)*. ACM, San Servolo Island, Venice, Italy, 205–214.

[7] L. Dalcin, M. Mortensen, and D. E. Keyes. 2018. Fast parallel multidimensional FFT using advanced MPI. *arXiv:1804.09536 [cs]* (April 2018). arXiv: 1804.09536.

[8] Oak Ridge Leadership Computing Facility. 2019. Summit system user guide. (2019). Retrieved Apr 10, 2019 from https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide/

[9] A. Gholami, J. Hill, D. Malhotra, and G. Biros. 2016. AccFFT: A library for distributed-memory FFT on CPU and GPU architectures. *CoRR* abs/1506.07933 (May 2016). arXiv:1506.07933 http://arxiv.org/abs/1506.07933

[10] T. Ishihara, K. Morishita, M. Yokokawa, A. Uno, and Y. Kaneda. 2016. Energy spectrum in high-resolution direct numerical simulation of turbulence. *Phys. Rev. Fluids* 1 (Dec. 2016), 082403.

[11] M. Lee, N. Malaya, and R. D. Moser. 2013. Petascale direct numerical simulation of turbulent channel flow up to 786K cores. In *International Conference for High Performance Computing, Networking and Storage Analysis (SC), Denver, CO*. ACM, New York, NY, USA, 61:1–61:11.

[12] P. D. Mininni, D. Rosenberg, R. Reddy, and A. Pouquet. 2011. A hybrid MPI-OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence. *Parallel Comput.* 37, 6 (June-July 2011), 316 – 326.

[13] Nvidia. 2019. The NVIDIA Visual Profiler. (2019). Retrieved April 10, 2019 from https://developer.nvidia.com/nvidia-visual-profiler

[14] D. Pekurovsky. 2012. P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions. *Siam. J. Sci. Comput.* 34 (Aug. 2012), C192–C209.

[15] Steven J Plimpton. 2017. *FFTs for (mostly) Particle Codes within the DOE Exascale Computing Project*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[16] S. B. Pope. 2000. *Turbulent Flows*. Cambridge University Press, Cambridge, UK.

[17] R. S. Rogallo. 1981. Numerical experiments in homogeneous turbulence. *NASA Tech. Memo. 81315*, NASA Ames Research Center. (Sept. 1981).

[18] G. Ruetsch and M. Fatica. 2014. *CUDA Fortran for Scientists and Engineers*. Morgan Kaufmann Publishers.

[19] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and P. S. Crozier. 2011. The Development of Mellanox/NVIDIA GPUDirect over

InfiniBand–a New Model for GPU to GPU Communications. *Comput. Sci.* 26, 3-4 (Jun 2011), 267–273.

[20] IBM POWER9 NPU team. 2018. Functionality and performance of NVLink with IBM POWER9 processors. *IBM Journal of Research and Development* 62, 4/5 (July 2018), 9:1–9:10. https://doi.org/10.1147/JRD.2018.2846978

[21] S. Vetter, A.B. Caldeira, and IBM Redbooks. 2018. *IBM Power System AC922 Introduction and Technical Overview*. IBM Redbooks.

[22] T. Watanabe, J. J. Riley, S. M. de Bruyn Kops, P. J. Diamessis, and Q. Zhou. 2016. Turbulent/non-turbulent interfaces in wakes in stably stratified fluids. *J. Fluid Mech.* 797 (June 2016), R1.

[23] P. K. Yeung, X. M. Zhai, and K. R. Sreenivasan. 2015. Extreme events in computational turbulence. *Proc. Nat. Acad. Sci.* 112 (Oct. 2015), 12633–12638.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran a Fourier pseudo-spectral simulation code on Summit using the IBM XL compiler v16.1.1-1 and IBM Spectrum MPI v 10.2.0.11. CUDA library v9.2.148 and FFTW library v3.3.8 were used to collect the data reported in the paper.

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

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:

NA

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: System name-Summit ; IBM AC922 nodes ; POWER9 CPUs ; NVIDIA VOLTA 100 GPUs ; IBM SpectrumScale file system ; Dual rail EDR InfiniBand network

Operating systems and versions: Red Hat Enterprise Linux Server 7.5 (Maipo) running Linux kernel 4.14.0-49.18.1

Compilers and versions: IBM XL compiler version 16.1.1-1

Libraries and versions: CUDA version 9.2.148 ; Spectrum-mpi version 10.2.0.11-20190201 ; FFTW version 3.3.8

Key algorithms: Fourier pseudo-spectral algorithm

Input datasets and versions: Self generated

Paper Modifications: No modifications were made to the hardware or Software. The Summit machine was used as is.

Output from scripts that gathers execution environment information.

```
LMOD_FAMILY_COMPILER_VERSION=16.1.1-1
MANPATH=/autofs/nccs-svm1_sw/summit/.swci/1-compute/
↪ opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1
↪ -1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaqbj6nnc/sh
↪ are/man:/autofs/nccs-svm1_sw/summit/.swci/0-core
↪ /opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.
↪ 5/vim-7.4.2367-dhtu3xylmvrjytqpllknptgkiepkvafx/
↪ share/man:/autofs/nccs-svm1_sw/summit/.swci/0-co
↪ re/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.
↪ 8.5/tmux-2.2-z2cgytxdo3rzw643uj2fiwp7iwqbbbwp/sh
↪ are/man:/sw/sources/hpss/man:/autofs/nccs-svm1_s
↪ w/summit/.swci/1-compute/opt/spack/20180914/linu
↪ x-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi-10.2.0.
↪ 11-20190201-6qypd6rixwrkcyd5gnijoacjqxrtblzk/sha
↪ re/man:/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/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmod/lmod
↪ /share/man:/opt/ibm/spectrumcomputing/lsf/10.1/m
↪ an::
XALT_ETC_DIR=/sw/summit/xalt/1.1.3/etc
XDG_SESSION_ID=53
HOSTNAME=login1
_ModuleTable003_=Y3RpdmUiLFsidXNlck5hbWUiXT0iZmZ0dyI
↪ sfSxoc2k9e1siZm4iXT0iL3N3L3N1bW1pdC9tb2R1bGVmaWx
↪ lcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0bGUvQ29yZS9oc2k
↪ vNS4wLjIucDUubHVhIixbImZ1bGxOYW1lIl09ImhzaS81LjA
↪ uMi5wNSIsWyJsb2FkT3JkZXIiXT0zLHByb3BUPXt9LFsic3R
↪ hY2tEZXB0aCJdPTEsWyJzdGF0dXMiXT0iYWN0aXZlIixbInV
↪ zZXJOYW1lIl09ImhzaSIsfSxbImxzZi10b29scyJdPXtbImZ
↪ uIl09Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW5
↪ 1eC1yaGVsNy1wcGM2NGxlL0NvcmUvbHNmLXRvb2xzLzIuMC5
↪ sdWEiLFsiZnVsbE5hbWUiXT0ibHNmLXRvb2xzLzIuMCIsWyJ
↪ sb2FkT3JkZXIiXT01LHByb3BUPXt9LFsic3RhY2tEZXB0
_ModuleTable009_=cy9Db3JlOi9zdy9zdW1taXQvbG1vZC83Ljc
↪ uMTAvcmhlbDcuM19nbnU0LjguNS9sbW9kL2xtb2QvbW9kdWx
↪ lZmlsZXMvQ29yZSIsfQ==
SHELL=/bin/bash
TERM=screen-256color
HISTSIZE=100000
LMOD_SYSTEM_DEFAULT_MODULES=DefApps
MODULEPATH_ROOT=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.
↪ 8.5/modulefiles
SSH_CLIENT=128.61.185.168 55666 22
LIBRARY_PATH=/sw/summit/cuda/9.2.148/lib64:/autofs/n
↪ ccs-svm1_sw/summit/.swci/1-compute/opt/spack/201
↪ 80914/linux-rhel7-ppc64le/xl-16.1.1-1/fftw-3.3.8
↪ -vuwn274gfobnmpcr6d3bbualaqbj6nnc/lib:/autofs/ncc
↪ s-svm1_sw/summit/.swci/1-compute/opt/spack/20180
↪ 914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi
↪ -10.2.0.11-20190201-6qypd6rixwrkcyd5gnijoacjqxrtb
↪ lzk/lib
PAMI_IBV_ADAPTER_AFFINITY=1
```

```
LMOD_PKG=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod
LSF_SERVERDIR=/opt/ibm/spectrumcomputing/lsf/10.1/li
↪ nux3.10-glibc2.17-ppc64le-csm/etc
OLCF_XL_ROOT=/sw/summit/xl/16.1.1-1
PAMI_ENABLE_STRIPING=0
LMOD_VERSION=7.7.10
OMPI_FC=/sw/summit/xl/16.1.1-1/xlf/16.1.1/bin/xlf200
↪ 8_r
OLCF_XLSMP_ROOT=/sw/summit/xl/16.1.1-1/xlsmp/5.1.1
SSH_TTY=/dev/pts/2
_ModuleTable007_=cGM2NGxlL3NwZWN0cnVtLW1waS8xMC4yLjA
↪ uMTEtMjAxOTAyMDEtNnF5cGQ2ci94bC8xNi4xLjEtMSIsIi9
↪ zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGV
↪ sNy1wcGM2NGxlL3hsLzE2LjEuMS0xIiwiL3N3L3N1bW1pdC9
↪ tb2R1bGVmaWxlcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0bGU
↪ vQ29yZSIsIi9zdy9zdW1taXQvbW9kdWxlZmlsZXMvY29yZSI
↪ sIi9zdy9zdW1taXQvbG1vZC83LjcuMTAvcmhlbDcuM19nbnU
↪ 0LjguNS9tb2R1bGVmaWxlcy9MaW51eCIsIi9zdy9zdW1taXQ
↪ vbG1vZC83LjcuMTAvcmhlbDcuM19nbnU0LjguNS9tb2R1bGV
↪ maWxlcy9Db3JlIiwiL3N3L3N1bW1pdC9sbW9kLzcuNy4xMC9
↪ yaGVsNy4zX2dudTQuOC41L2xtb2QvbG1vZC9tb2R1bGVm
__LMOD_REF_COUNT_CMAKE_PREFIX_PATH=/sw/summit/cuda/9
↪ .2.148:1;/autofs/nccs-svm1_sw/summit/.swci/1-com
↪ pute/opt/spack/20180914/linux-rhel7-ppc64le/xl-1
↪ 6.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaqbj6
↪ nnc:1;/autofs/nccs-svm1_sw/summit/.swci/0-core/o
↪ pt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/
↪ vim-7.4.2367-dhtu3xylmvrjytqpllknptgkiepkvafx:1;
↪ /autofs/nccs-svm1_sw/summit/.swci/0-core/opt/spa
↪ ck/20180914/linux-rhel7-ppc64le/gcc-4.8.5/tmux-2
↪ .2-z2cgytxdo3rzw643uj2fiwp7iwqbbbwp:1;/autofs/nc
↪ cs-svm1_sw/summit/.swci/1-compute/opt/spack/2018
↪ 0914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mp
↪ i-10.2.0.11-20190201-6qypd6rixwrkcyd5gnijoacjqxr
↪ tblzk:1
__LMOD_REF_COUNT_LOADEDMODULES=xl/16.1.1-1:1;spectru
↪ m-mpi/10.2.0.11-20190201:1;hsi/5.0.2.p5:1;xalt/1
↪ .1.3:1;lsf-tools/2.0:1;DefApps:1;tmux/2.2:1;vim/
↪ 7.4.2367:1;fftw/3.3.8:1;cuda/9.2.148:1
LSF_LIBDIR=/opt/ibm/spectrumcomputing/lsf/10.1/linux
↪ 3.10-glibc2.17-ppc64le-csm/lib
OPAL_PREFIX=/autofs/nccs-svm1_sw/summit/.swci/1-comp
↪ ute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16
↪ .1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6rix
↪ wrkcyd5gnijoacjqxrtblzk
USER=USER
LD_LIBRARY_PATH=/sw/summit/cuda/9.2.148/lib64:/autof
↪ s/nccs-svm1_sw/summit/.swci/1-compute/opt/spack/
↪ 20180914/linux-rhel7-ppc64le/xl-16.1.1-1/fftw-3.
↪ 3.8-vuwn274gfobnmpcr6d3bbualaqbj6nnc/lib:/autofs
↪ /nccs-svm1_sw/summit/.swci/1-compute/opt/spack/2
↪ 0180914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum
↪ -mpi-10.2.0.11-20190201-6qypd6rixwrkcyd5gnijoacjq
↪ xrtblzk/lib:/sw/summit/xl/16.1.1-1/xlsmp/5.1.1/l
↪ ib:/sw/summit/xl/16.1.1-1/xlmass/9.1.1/lib:/sw/s
↪ ummit/xl/16.1.1-1/xlC/16.1.1/lib:/sw/summit/xl/1
↪ 6.1.1-1/xlf/16.1.1/lib:/sw/summit/xl/16.1.1-1/li
↪ b:/opt/ibm/spectrumcomputing/lsf/10.1/linux3.10-
↪ glibc2.17-ppc64le-csm/lib:/opt/ibm/spectrum_mpi/
↪ jsm_pmix/lib
LMOD_sys=Linux
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:
PAMI_IBV_ENABLE_OOO_AR=1
LMOD_MPI_NAME=spectrum-mpi
```

```
CPATH=/sw/summit/cuda/9.2.148/include:/autofs/nccs-s
↪ vm1_sw/summit/.swci/1-compute/opt/spack/20180914
↪ /linux-rhel7-ppc64le/xl-16.1.1-1/fftw-3.3.8-vuwn
↪ 274gfobnmpcr6d3bbualaqbj6nnc/include:/autofs/ncc
↪ s-svm1_sw/summit/.swci/1-compute/opt/spack/20180
↪ 914/linux-rhel7-ppc64le/xl-16.1.1-1/spectrum-mpi
↪ -10.2.0.11-20190201-6qypd6rixwrkcyd5gnijoacjqxrtb
↪ lzk/include
_ModuleTable004_=aCJdPTEsWyJzdGF0dXMiXT0iYWN0aXZlIix
↪ bInVzZXJOYW1lIl09ImxzZi10b29scyIsfSxbInNwZWN0cnV
↪ tLW1waSJdPXtbImZuIl09Ii9zdy9zdW1taXQvbW9kdWxlZml
↪ sZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL3hsLzE2LjE
↪ uMS0xL3NwZWN0cnVtLW1waS8xMC4yLjAuMTEtMjAxOTAyMDE
↪ ubHVhIixbImZ1bGxOYW1lIl09InNwZWN0cnVtLW1waS8xMC4
↪ yLjAuMTEtMjAxOTAyMDEiLFsibG9hZE9yZGVyIl09Mixwcm9
↪ wVD17fSxbInN0YWNrRGVwdGgiXT0xLFsic3RhdHVzIl09ImF
↪ jdGl2ZSIsWyJ1c2VyTmFtZSJdPSJzcGVjdHJ1bS1tcGkiLH0
↪ sdG11eD17WyJmbiJdPSIvc3cvc3VtbWl0L21vZHVsZWZpbGV
↪ zL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS9Db3JlL3Rt
__LMOD_REF_COUNT__LMFILES_=/sw/summit/modulefiles/si
↪ te/linux-rhel7-ppc64le/Core/xl/16.1.1-1.lua:1;/s
↪ w/summit/modulefiles/site/linux-rhel7-ppc64le/xl
↪ /16.1.1-1/spectrum-mpi/10.2.0.11-20190201.lua:1;
↪ /sw/summit/modulefiles/site/linux-rhel7-ppc64le/
↪ Core/hsi/5.0.2.p5.lua:1;/sw/summit/modulefiles/s
↪ ite/linux-rhel7-ppc64le/Core/xalt/1.1.3.lua:1;/s
↪ w/summit/modulefiles/site/linux-rhel7-ppc64le/Co
↪ re/lsf-tools/2.0.lua:1;/sw/summit/modulefiles/si
↪ te/linux-rhel7-ppc64le/Core/DefApps.lua:1;/sw/su
↪ mmit/modulefiles/site/linux-rhel7-ppc64le/Core/t
↪ mux/2.2.lua:1;/sw/summit/modulefiles/site/linux-
↪ rhel7-ppc64le/Core/vim/7.4.2367.lua:1;/autofs/nc
↪ cs-svm1_sw/summit/modulefiles/site/linux-rhel7-p
↪ pc64le/spectrum-mpi/10.2.0.11-20190201-6qypd6r/x
↪ l/16.1.1-1/fftw/3.3.8.lua:1;/sw/summit/modulefil
↪ es/site/linux-rhel7-ppc64le/xl/16.1.1-1/cuda/9.2
↪ .148.lua:1
OLCF_LMOD_ROOT=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8
↪ .5
OLCF_XLMASS_ROOT=/sw/summit/xl/16.1.1-1/xlmass/9.1.1
PROJWORK=/gpfs/alpine/proj-shared
TMUX=/tmp/tmux-13765/default,82078,0
LMOD_FAMILY_MPI_VERSION=10.2.0.11-20190201
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
MAIL=/var/spool/mail/USER
PATH=/sw/sources/lsf-tools/2.0/summit/bin:/sw/summit
↪ /xalt/1.1.3/bin:/sw/summit/cuda/9.2.148/bin:/aut
↪ ofs/nccs-svm1_sw/summit/.swci/1-compute/opt/spac
↪ k/20180914/linux-rhel7-ppc64le/xl-16.1.1-1/fftw-
↪ 3.3.8-vuwn274gfobnmpcr6d3bbualaqbj6nnc/bin:/ccs/
↪ home/USER/scripts/Archival-Scripts:/ccs/home/USE
↪ R/scripts:/autofs/nccs-svm1_sw/summit/.swci/0-co
↪ re/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.
↪ 8.5/vim-7.4.2367-dhtu3xylmvrjytqpllknptgkiepkvaf
↪ x/bin:/autofs/nccs-svm1_sw/summit/.swci/0-core/o
↪ pt/spack/20180914/linux-rhel7-ppc64le/gcc-4.8.5/
↪ tmux-2.2-z2cgytxdo3rzw643uj2fiwp7iwqbbbwp/bin:/u
↪ sr/bin:/usr/sbin:/opt/ibm/csm/bin:/sw/sources/hp
↪ ss/bin:/autofs/nccs-svm1_sw/summit/.swci/1-compu
↪ te/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.
↪ 1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6rixw
↪ rkcyd5gnijoacjqxrtblzk/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/linux
↪ 3.10-glibc2.17-ppc64le-csm/etc:/opt/ibm/spectrum
↪ computing/lsf/10.1/linux3.10-glibc2.17-ppc64le-c
↪ sm/bin:/usr/local/bin:/usr/local/sbin:/opt/ibm/f
↪ lightlog/bin:/opt/ibutils/bin:/opt/ibm/spectrum_
↪ mpi/jsm_pmix/bin:/opt/puppetlabs/bin:/usr/lpp/mm
↪ fs/bin
_ModuleTable001_=X01vZHVsZVRhYmxlXz17WyJNVHZlcnNpb24
↪ iXT0zLFsiY19yZWJ1aWxkVGltZSJdPWZhbHNlLFsiY19zaG9
↪ ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9e30sZmFtaWx5PXtbImN
↪ vbXBpbGVyIl09InhsIixbIm1waSJdPSJzcGVjdHJ1bS1tcGk
↪ iLH0sbVQ9e0RlZkFwcHM9e1siZm4iXT0iL3N3L3N1bW1pdC9
↪ tb2R1bGVmaWxlcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0bGU
↪ vQ29yZS9EZWZBcHBzLmx1YSIsWyJmdWxsTmFtZSJdPSJEZWZ
↪ BcHBzIixbImxvYWRPcmRlciJdPTYscHJvcFQ9e30sWyJzdGF
↪ ja0RlcHRoIl09MCxbInN0YXR1cyJdPSJhY3RpdmUiLFsidXN
↪ lck5hbWUiXT0iRGVmQXBwcyIsfSxjdWRhPXtbImZuIl09Ii9
↪ zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1y
__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
PAMI_IBV_QP_SERVICE_LEVEL=8
_=/usr/bin/env
OLCF_XLC_ROOT=/sw/summit/xl/16.1.1-1/xlC/16.1.1
OPAL_LIBDIR=/autofs/nccs-svm1_sw/summit/.swci/1-comp
↪ ute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16
↪ .1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6rix
↪ wrkcyd5gnijoacjqxrtblzk/lib
OMPI_DIR=/autofs/nccs-svm1_sw/summit/.swci/1-compute
↪ /opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.
↪ 1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6rixwrk
↪ cyd5gnijoacjqxrtblzk
PWD=/gpfs/alpine/proj-shared/tur120/USER/PSDNS_HOMO_
↪ OMP/2019-04-10-Performance
OLCF_VIM_ROOT=/autofs/nccs-svm1_sw/summit/.swci/0-co
↪ re/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4.
↪ 8.5/vim-7.4.2367-dhtu3xylmvrjytqpllknptgkiepkvafx
```

```
_LMFILES_=/sw/summit/modulefiles/site/linux-rhel7-pp
↪ c64le/Core/xl/16.1.1-1.lua:/sw/summit/modulefile
↪ s/site/linux-rhel7-ppc64le/xl/16.1.1-1/spectrum-
↪ mpi/10.2.0.11-20190201.lua:/sw/summit/modulefile
↪ s/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
↪ /DefApps.lua:/sw/summit/modulefiles/site/linux-r
↪ hel7-ppc64le/Core/tmux/2.2.lua:/sw/summit/module
↪ files/site/linux-rhel7-ppc64le/Core/vim/7.4.2367
↪ .lua:/autofs/nccs-svm1_sw/summit/modulefiles/sit
↪ e/linux-rhel7-ppc64le/spectrum-mpi/10.2.0.11-201
↪ 90201-6qypd6r/xl/16.1.1-1/fftw/3.3.8.lua:/sw/sum
↪ mit/modulefiles/site/linux-rhel7-ppc64le/xl/16.1
↪ .1-1/cuda/9.2.148.lua
EDITOR=/usr/bin/vim
OLCF_MODULEPATH_ROOT=/sw/summit/modulefiles
OLCF_CUDA_ROOT=/sw/summit/cuda/9.2.148
__LMOD_REF_COUNT_PYTHONPATH=/sw/summit/xalt/1.1.3/si
↪ te:1;/sw/summit/xalt/1.1.3/libexec:1
MODULEPATH=/autofs/nccs-svm1_sw/summit/modulefiles/s
↪ ite/linux-rhel7-ppc64le/spectrum-mpi/10.2.0.11-2
↪ 0190201-6qypd6r/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
LMOD_SYSTEM_NAME=summit
LOADEDMODULES=xl/16.1.1-1:spectrum-mpi/10.2.0.11-201
↪ 90201:hsi/5.0.2.p5:xalt/1.1.3:lsf-tools/2.0:DefA
↪ pps:tmux/2.2:vim/7.4.2367:fftw/3.3.8:cuda/9.2.148
_ModuleTable_Sz_=8
TMUX_PANE=%3
OLCF_XLF_ROOT=/sw/summit/xl/16.1.1-1/xlf/16.1.1
LMOD_CMD=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod/libexec/lmod
_ModuleTable005_=dXgvMi4yLmx1YSIsWyJmdWxsTmFtZSJdPSJ
↪ 0bXV4LzIuMiIsWyJsb2FkT3JkZXIiXT03LHByb3BUPXt9LFs
↪ ic3RhY2tEZXB0aCJdPTAsWyJzdGF0dXMiXT0iYWN0aXZlIix
↪ bInVzZXJOYW1lIl09InRtdXgiLH0sdmltPXtbImZuIl09Ii9
↪ zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGV
↪ sNy1wcGM2NGxlL0NvcmUvdmltLzcuNC4yMzY3Lmx1YSIsWyJ
↪ mdWxsTmFtZSJdPSJ2aW0vNy40LjIzNjciLFsibG9hZE9yZGV
↪ yIl09OCxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0wLFsic3R
↪ hdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJ2aW0iLH0
↪ seGFsdD17WyJmbiJdPSIvc3cvc3VtbWl0L21vZHVsZWZpbGV
↪ zL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS9Db3JlL3hh
LSF_BINDIR=/opt/ibm/spectrumcomputing/lsf/10.1/linux
↪ 3.10-glibc2.17-ppc64le-csm/bin
HISTCONTROL=ignoredups:ignorespace
WORLDWORK=/gpfs/alpine/world-shared
HOME=/ccs/home/USER
MEMBERWORK=/gpfs/alpine/scratch/USER
SHLVL=3
OMPI_CC=/sw/summit/xl/16.1.1-1/xlC/16.1.1/bin/xlc_r
__LMOD_REF_COUNT_PATH=/sw/sources/lsf-tools/2.0/summ
↪ it/bin:3;/sw/summit/xalt/1.1.3/bin:1;/sw/summit/
↪ cuda/9.2.148/bin:1;/autofs/nccs-svm1_sw/summit/.
↪ swci/1-compute/opt/spack/20180914/linux-rhel7-pp
↪ c64le/xl-16.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3
↪ bbualaqbj6nnc/bin:1;/ccs/home/USER/scripts/Archi
↪ val-Scripts:2;/ccs/home/USER/scripts:2;/autofs/n
↪ ccs-svm1_sw/summit/.swci/0-core/opt/spack/201809
↪ 14/linux-rhel7-ppc64le/gcc-4.8.5/vim-7.4.2367-dh
↪ tu3xylmvrjytqpllknptgkiepkvafx/bin:1;/autofs/ncc
↪ s-svm1_sw/summit/.swci/0-core/opt/spack/20180914
↪ /linux-rhel7-ppc64le/gcc-4.8.5/tmux-2.2-z2cgytxd
↪ o3rzw643uj2fiwp7iwqbbbwp/bin:1;/usr/bin:3;/usr/s
↪ bin:3;/opt/ibm/csm/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/s
↪ pectrum-mpi-10.2.0.11-20190201-6qypd6rixwrkcyd5g
↪ nijoacjqxrtblzk/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/spectru
↪ mcomputing/lsf/10.1/linux3.10-glibc2.17-ppc64le-
↪ csm/bin:1;/usr/local/bin:1;/usr/local/sbin:1;/op
↪ t/ibm/flightlog/bin:1;/opt/ibutils/bin:1;/opt/ib
↪ m/spectrum_mpi/jsm_pmix/bin:1;/opt/puppetlabs/bi
↪ n:1;/usr/lpp/mmfs/bin:1
__LMOD_REF_COUNT_CPATH=/sw/summit/cuda/9.2.148/inclu
↪ de:1;/autofs/nccs-svm1_sw/summit/.swci/1-compute
↪ /opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.
↪ 1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaqbj6nnc/
↪ include:1;/autofs/nccs-svm1_sw/summit/.swci/1-co
↪ mpute/opt/spack/20180914/linux-rhel7-ppc64le/xl-
↪ 16.1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6r
↪ ixwrkcyd5gnijoacjqxrtblzk/include:1
BINARY_TYPE_HPC=
_ModuleTable002_=aGVsNy1wcGM2NGxlL3hsLzE2LjEuMS0xL2N
↪ 1ZGEvOS4yLjE0OC5sdWEiLFsiZnVsbE5hbWUiXT0iY3VkYS8
↪ 5LjIuMTQ4IixbImxvYWRPcmRlciJdPTEwLHByb3BUPXt9LFs
↪ ic3RhY2tEZXB0aCJdPTAsWyJzdGF0dXMiXT0iYWN0aXZlIix
↪ bInVzZXJOYW1lIl09ImN1ZGEiLH0sZmZ0dz17WyJmbiJdPSI
↪ vYXV0b2ZzL25jY3Mtc3ZtMV9zdy9zdW1taXQvbW9kdWxlZml
↪ sZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL3NwZWN0cnV
↪ tLW1waS8xMC4yLjAuMTEtMjAxOTAyMDEtNnF5cGQ2ci94bC8
↪ xNi4xLjEtMS9mZnR3LzMuMy44Lmx1YSIsWyJmdWxsTmFtZSJ
↪ dPSJmZnR3LzMuMy44IixbImxvYWRPcmRlciJdPTkscHJvcFQ
↪ 9e30sWyJzdGFja0RlcHRoIl09MCxbInN0YXR1cyJdPSJh
SHOST=login1
BASH_ENV=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪ d/lmod/init/bash
OLCF_HSI_ROOT=/sw/sources/hpss
```

```
OLCF_TMUX_ROOT=/autofs/nccs-svm1_sw/summit/.swci/0-c
↪  ore/opt/spack/20180914/linux-rhel7-ppc64le/gcc-4
↪  .8.5/tmux-2.2-z2cgytxdo3rzw643uj2fiwp7iwqbbbwp
_ModuleTable008_=aWxlcy9Db3JlIix9LFsic3lzdGVtQmFzZU1
↪  QQVRIIl09Ii9zdy9zdW1taXQvbG1vZC83LjcuMTAvcmhlbDc
↪  uM19nbnU0LjguNS9tb2R1bGVmaWxlcy9MaW51eDovc3cvc3V
↪  tbWl0L2xtb2QvNy43LjEwL3JoZWw3LjNfZ251NC44LjUvbW9
↪  kdWxlZmlsZXMvQ29yZTovc3cvc3VtbWl0L2xtb2QvNy43LjE
↪  wL3JoZWw3LjNfZ251NC44LjUvbG1vZC9sbW9kL21vZHVsZWZ
↪  pbGVzL0NvcmUiLH0=
LESS= -R -R
LOGNAME=USER
MPI_ROOT=/autofs/nccs-svm1_sw/summit/.swci/1-compute
↪  /opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.
↪  1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6rixwrk
↪  cyd5gnijoacjqxrtblzk
PAMI_IBV_ENABLE_DCT=1
PYTHONPATH=/sw/summit/xalt/1.1.3/site:/sw/summit/xal
↪  t/1.1.3/libexec
CVS_RSH=ssh
OLCF_SPECTRUM_MPI_ROOT=/autofs/nccs-svm1_sw/summit/.
↪  swci/1-compute/opt/spack/20180914/linux-rhel7-pp
↪  c64le/xl-16.1.1-1/spectrum-mpi-10.2.0.11-2019020
↪  1-6qypd6rixwrkcyd5gnijoacjqxrtblzk
SSH_CONNECTION=128.61.185.168 55666 128.219.134.71 22
XLSF_UIDDIR=/opt/ibm/spectrumcomputing/lsf/10.1/linu
↪  x3.10-glibc2.17-ppc64le-csm/lib/uid
MODULESHOME=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/
↪  lmod/lmod
__LMOD_REF_COUNT_LIBRARY_PATH=/sw/summit/cuda/9.2.14
↪  8/lib64:1;/autofs/nccs-svm1_sw/summit/.swci/1-co
↪  mpute/opt/spack/20180914/linux-rhel7-ppc64le/xl-
↪  16.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaqbj
↪  6nnc/lib:1;/autofs/nccs-svm1_sw/summit/.swci/1-c
↪  ompute/opt/spack/20180914/linux-rhel7-ppc64le/xl
↪  -16.1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6r
↪  ixwrkcyd5gnijoacjqxrtblzk/lib:1
LESSOPEN=||/usr/bin/lesspipe.sh %s
LMOD_SETTARG_FULL_SUPPORT=no
OMPI_CXX=/sw/summit/xl/16.1.1-1/xlC/16.1.1/bin/xlc++
↪  _r
PKG_CONFIG_PATH=/autofs/nccs-svm1_sw/summit/.swci/1-
↪  compute/opt/spack/20180914/linux-rhel7-ppc64le/x
↪  l-16.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaq
↪  bj6nnc/lib/pkgconfig
__LMOD_REF_COUNT_LD_LIBRARY_PATH=/sw/summit/cuda/9.2
↪  .148/lib64:1;/autofs/nccs-svm1_sw/summit/.swci/1
↪  -compute/opt/spack/20180914/linux-rhel7-ppc64le/x
↪  l-16.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaq
↪  bj6nnc/lib:1;/autofs/nccs-svm1_sw/summit/.swci/1
↪  -compute/opt/spack/20180914/linux-rhel7-ppc64le/x
↪  l-16.1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd
↪  6rixwrkcyd5gnijoacjqxrtblzk/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/summit/xl/16.1.1-1/xlf/16.1.1/l
↪  ib:1;/sw/summit/xl/16.1.1-1/lib:1;/opt/ibm/spect
↪  rumcomputing/lsf/10.1/linux3.10-glibc2.17-ppc64l
↪  e-csm/lib:1;/opt/ibm/spectrum_mpi/jsm_pmix/lib:1
LMOD_MPI_VERSION=10.2.0.11-20190201-6qypd6r
OMPI_MCA_io=romio321
__Init_Default_Modules=1
LMOD_FAMILY_COMPILER=xl
OLCF_FFTW_ROOT=/autofs/nccs-svm1_sw/summit/.swci/1-c
↪  ompute/opt/spack/20180914/linux-rhel7-ppc64le/xl
↪  -16.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6d3bbualaqbj
↪  6nnc
CMAKE_PREFIX_PATH=/sw/summit/cuda/9.2.148:/autofs/nc
↪  cs-svm1_sw/summit/.swci/1-compute/opt/spack/2018
↪  0914/linux-rhel7-ppc64le/xl-16.1.1-1/fftw-3.3.8-
↪  vuwn274gfobnmpcr6d3bbualaqbj6nnc:/autofs/nccs-sv
↪  m1_sw/summit/.swci/0-core/opt/spack/20180914/lin
↪  ux-rhel7-ppc64le/gcc-4.8.5/vim-7.4.2367-dhtu3xyl
↪  mvrjytqpllknptgkiepkvafx:/autofs/nccs-svm1_sw/su
↪  mmit/.swci/0-core/opt/spack/20180914/linux-rhel7
↪  -ppc64le/gcc-4.8.5/tmux-2.2-z2cgytxdo3rzw643uj2fi
↪  wp7iwqbbbwp:/autofs/nccs-svm1_sw/summit/.swci/1-
↪  compute/opt/spack/20180914/linux-rhel7-ppc64le/x
↪  l-16.1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd
↪  6rixwrkcyd5gnijoacjqxrtblzk
XALT_OLCF=1
XDG_RUNTIME_DIR=/run/user/13765
XL_LINKER=/sw/summit/xalt/1.1.3/bin/ld
__LMOD_REF_COUNT_PKG_CONFIG_PATH=/autofs/nccs-svm1_s
↪  w/summit/.swci/1-compute/opt/spack/20180914/linu
↪  x-rhel7-ppc64le/xl-16.1.1-1/fftw-3.3.8-vuwn274gf
↪  obnmpcr6d3bbualaqbj6nnc/lib/pkgconfig:1
LMOD_DIR=/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪  d/lmod/libexec
LSF_ENVDIR=/opt/ibm/spectrumcomputing/lsf/conf
_ModuleTable006_=bHQvMS4xLjMubHVhIixbImZ1bGxOYW1lIl0
↪  9InhhbHQvMS4xLjMiLFsibG9hZE9yZGVyIl09NCxwcm9wVD1
↪  7fSxbInN0YWNrRGVwdGgiXT0xLFsic3RhdHVzIl09ImFjdGl
↪  2ZSIsWyJ1c2VyTmFtZSJdPSJ4YWx0Iix9LHhsPXtbImZuIl0
↪  9Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1
↪  yaGVsNy1wcGM2NGxlL0NvcmUveGwvMTYuMS4xLTEubHVhIix
↪  bImZ1bGxOYW1lIl09InhsLzE2LjEuMS0xIixbImxvYWRPcmR
↪  lciJdPTEscHJvcFQ9e30sWyJzdGFja0RlcHRoIl09MSxbInN
↪  0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5hbWUiXT0ieGwiLH0
↪  sfSxtcGF0aEE9eyIvYXV0b2ZzL25jY3Mtc3ZtMV9zdy9zdW1
↪  taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1w
```

```
__LMOD_REF_COUNT_MANPATH=/autofs/nccs-svm1_sw/summit
↪  /.swci/1-compute/opt/spack/20180914/linux-rhel7-
↪  ppc64le/xl-16.1.1-1/fftw-3.3.8-vuwn274gfobnmpcr6
↪  d3bbualaqbj6nnc/share/man:1;/autofs/nccs-svm1_sw
↪  /summit/.swci/0-core/opt/spack/20180914/linux-rh
↪  el7-ppc64le/gcc-4.8.5/vim-7.4.2367-dhtu3xylmvrjy
↪  tqpllknptgkiepkvafx/share/man:1;/autofs/nccs-svm
↪  1_sw/summit/.swci/0-core/opt/spack/20180914/linu
↪  x-rhel7-ppc64le/gcc-4.8.5/tmux-2.2-z2cgytxdo3rzw
↪  643uj2fiwp7iwqbbbwp/share/man:1;/sw/sources/hpss
↪  /man:1;/autofs/nccs-svm1_sw/summit/.swci/1-compu
↪  te/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.
↪  1.1-1/spectrum-mpi-10.2.0.11-20190201-6qypd6rixw
↪  rkcyd5gnijoacjqxrtblzk/share/man:1;/sw/summit/xl
↪  /16.1.1-1/xlC/16.1.1/man/en_US:1;/sw/summit/xl/1
↪  6.1.1-1/xlf/16.1.1/man/en_US:1;/sw/summit/lmod/7
↪  .7.10/rhel7.3_gnu4.8.5/lmod/lmod/share/man:1;/op
↪  t/ibm/spectrumcomputing/lsf/10.1/man:1
__LMOD_Priority_PATH=/sw/sources/lsf-tools/2.0/summi
↪  t/bin:-9999;/sw/summit/xalt/1.1.3/bin:-9999
LMOD_FAMILY_MPI=spectrum-mpi
BASH_FUNC_module()=() {  eval $($LMOD_CMD bash "$@")
↪  && eval $(${LMOD_SETTARG_CMD:-:} -s sh)
}
BASH_FUNC_ml()=() {  eval $($LMOD_DIR/ml_cmd "$@")
}
+ lsb_release -a
LSB Version:        :core-4.1-noarch:core-4.1-ppc64le
Distributor ID:        RedHatEnterpriseServer
Description:        Red Hat Enterprise Linux Server
↪  release 7.5 (Maipo)
Release:        7.5
Codename:        Maipo
+ uname -a
Linux login1 4.14.0-49.18.1.el7a.ppc64le #1 SMP Thu
↪  Nov 29 03:27:24 EST 2018 ppc64le ppc64le ppc64le
↪  GNU/Linux
+ lscpu
Architecture:          ppc64le
Byte Order:            Little Endian
CPU(s):                128
On-line CPU(s) list:   0-127
Thread(s) per core:    4
Core(s) per socket:    16
Socket(s):             2
NUMA node(s):          6
Model:                 2.1 (pvr 004e 1201)
Model name:            POWER9, altivec supported
CPU max MHz:           3800.0000
CPU min MHz:           2300.0000
L1d cache:             32K
L1i cache:             32K
L2 cache:              512K
L3 cache:              10240K
NUMA node0 CPU(s):     0-63
NUMA node8 CPU(s):     64-127
NUMA node252 CPU(s):
NUMA node253 CPU(s):
NUMA node254 CPU(s):
NUMA node255 CPU(s):
+ cat /proc/meminfo
MemTotal:       601183424 kB
MemFree:        12398080 kB
MemAvailable:   40824640 kB
Buffers:               0 kB
Cached:         50194176 kB
SwapCached:            0 kB
Active:         379617344 kB
Inactive:       24476800 kB
Active(anon):   371714560 kB
Inactive(anon):  3353152 kB
Active(file):    7902784 kB
Inactive(file): 21123648 kB
Unevictable:    16819840 kB
Mlocked:        16819840 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                64 kB
Writeback:             0 kB
AnonPages:      370375872 kB
Mapped:         157961664 kB
Shmem:          21175232 kB
Slab:            5219968 kB
SReclaimable:    2000960 kB
SUnreclaim:      3219008 kB
KernelStack:       80624 kB
PageTables:      1108864 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    300591680 kB
Committed_AS:   646581952 kB
VmallocTotal:   549755813888 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:   128 kB
AnonHugePages:   3276800 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:       26853376 kB
CmaFree:            6080 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
+ inxi -F -c0
/ccs/home/kiran92/scripts/SC19AD-script.sh: line 14:
↪  inxi: command not found
+ lsblk -a
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb       8:16   1  1.8T  0 disk
```

```
loop0     7:0    0        1 loop
sda       8:0    1  1.8T  0 disk
nvme0n1 259:0    0  1.5T  0 disk
+ lsscsi -s
/ccs/home/kiran92/scripts/SC19AD-script.sh: line 16:
↪  lsscsi: command not found
+ module list
++ /sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmod/lmod
↪  /libexec/lmod bash
↪  list

Currently Loaded Modules:
  1) xl/16.1.1-1   2) spectrum-mpi/10.2.0.11-20190201
  ↪   3) hsi/5.0.2.p5   4) xalt/1.1.3   5)
  ↪  lsf-tools/2.0   6) DefApps   7) tmux/2.2   8)
  ↪  vim/7.4.2367   9) fftw/3.3.8  10) cuda/9.2.148



+ eval 'MODULEPATH="/autofs/nccs-svm1_sw/summit/modu
↪  lefiles/site/linux-rhel7-ppc64le/spectrum-mpi/10
↪  .2.0.11-20190201-6qypd6r/xl/16.1.1-1:/sw/summit/
↪  modulefiles/site/linux-rhel7-ppc64le/xl/16.1.1-1
↪  :/sw/summit/modulefiles/site/linux-rhel7-ppc64le
↪  /Core:/sw/summit/modulefiles/core:/sw/summit/lmo
↪  d/7.7.10/rhel7.3_gnu4.8.5/modulefiles/Linux:/sw/
↪  summit/lmod/7.7.10/rhel7.3_gnu4.8.5/modulefiles/
↪  Core:/sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmo
↪  d/lmod/modulefiles/Core";'
export 'MODULEPATH;' '_ModuleTable001_="X01vZHVsZVRh
↪  YmxlXz17WyJNVHZlcnNpb24iXT0zLFsiY19yZWJ1aWxkVGlt
↪  ZSJdPWZhbHNlLFsiY19zaG9ydFRpbWUiXT1mYWxzZSxkZXB0
↪  aFQ9e30sZmFtaWx5PXtbImNvbXBpbGVyIl09InhsIixbIm1w
↪  aSJdPSJzcGVjdHJ1bS1tcGkiLH0sbVQ9e0RlZkFwcHM9e1si
↪  Zm4iXT0iL3N3L3N1bW1pdC9tb2R1bGVmaWxlcy9zaXRlL2xp
↪  bnV4LXJoZWw3LXBwYzY0bGUvQ29yZS9EZWZBcHBzLmx1YSIs
↪  WyJmdWxsTmFtZSJdPSJEZWZBcHBzIixbImxvYWRPcmRlciJd
↪  PTYscHJvcFQ9e30sWyJzdGFja0RlcHRoIl09MCxbInN0YXR1
↪  cyJdPSJhY3RpdmUiLFsidXNlck5hbWUiXT0iRGVmQXBwcyIs
↪  fSxjdWRhPXtbImZuIl09Ii9zdy9zdW1taXQvbW9kdWxlZmls
↪  ZXMvc2l0ZS9saW51eC1y";'
export '_ModuleTable001_;' '_ModuleTable002_="aGVsNy
↪  1wcGM2NGxlL3hsLzE2LjEuMS0xL2N1ZGEvOS4yLjE0OC5sdW
↪  EiLFsiZnVsbE5hbWUiXT0iY3VkYS85LjIuMTQ4IixbImxvYW
↪  RPcmRlciJdPTEwLHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPT
↪  AsWyJzdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09Im
↪  N1ZGEiLH0sZmZ0dz17WyJmbiJdPSIvYXV0b2ZzL25jY3Mtc3
↪  ZtMV9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC
↪  1yaGVsNy1wcGM2NGxlL3NwZWN0cnVtLW1waS8xMC4yLjAuMT
↪  EtMjAxOTAyMDEtNnF5cGQ2ci94bC8xNi4xLjEtMS9mZnR3Lz
↪  MuMy44Lmx1YSIsWyJmdWxsTmFtZSJdPSJmZnR3LzMuMy44Ii
↪  xbImxvYWRPcmRlciJdPTkscHJvcFQ9e30sWyJzdGFja0RlcH
↪  RoIl09MCxbInN0YXR1cyJdPSJh";'
export '_ModuleTable002_;' '_ModuleTable003_="Y3Rpdm
↪  UiLFsidXNlck5hbWUiXT0iZmZ0dyIsfSxoc2k9e1siZm4iXT
↪  0iL3N3L3N1bW1pdC9tb2R1bGVmaWxlcy9zaXRlL2xpbnV4LX
↪  JoZWw3LXBwYzY0bGUvQ29yZS9oc2kvNS4wLjIucDUubHVhIi
↪  xbImZ1bGxOYW1lIl09ImhzaS81LjAuMi5wNSIsWyJsb2FkT3
↪  JkZXIiXT0zLHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTEsWy
↪  JzdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09ImhzaS
↪  IsfSxbImxzZi10b29scyJdPXtbImZuIl09Ii9zdy9zdW1taX
↪  QvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NG
↪  xlL0NvcmUvbHNmLXRvb2xzLzIuMC5sdWEiLFsiZnVsbE5hbW
↪  UiXT0ibHNmLXRvb2xzLzIuMCIsWyJsb2FkT3JkZXIiXT01LH
↪  Byb3BUPXt9LFsic3RhY2tEZXB0";'
export '_ModuleTable003_;'
 '_ModuleTable004_="aCJdPTEsWyJzdGF0dXMiXT0iYWN0aXZl
 ↪  IixbInVzZXJOYW1lIl09ImxzZi10b29scyIsfSxbInNwZWN
 ↪  0cnVtLW1waSJdPXtbImZuIl09Ii9zdy9zdW1taXQvbW9kdW
 ↪  xlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL3hsL
 ↪  zE2LjEuMS0xL3NwZWN0cnVtLW1waS8xMC4yLjAuMTEtMjAx
 ↪  OTAyMDEubHVhIixbImZ1bGxOYW1lIl09InNwZWN0cnVtLW1
 ↪  waS8xMC4yLjAuMTEtMjAxOTAyMDEiLFsibG9hZE9yZGVyIl
 ↪  09Mixwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0xLFsic3Rhd
 ↪  HVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJzcGVjdHJ1
 ↪  bS1tcGkiLH0sdG11eD17WyJmbiJdPSIvc3cvc3VtbWl0L21
 ↪  vZHVsZWZpbGVzL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS
 ↪  9Db3JlL3Rt";'
export '_ModuleTable004_;' '_ModuleTable005_="dXgvMi
↪  4yLmx1YSIsWyJmdWxsTmFtZSJdPSJ0bXV4LzIuMiIsWyJsb2
↪  FkT3JkZXIiXT03LHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPT
↪  AsWyJzdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09In
↪  RtdXgiLH0sdmltPXtbImZuIl09Ii9zdy9zdW1taXQvbW9kdW
↪  xlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL0Nvcm
↪  UvdmltLzcuNC4yMzY3Lmx1YSIsWyJmdWxsTmFtZSJdPSJ2aW
↪  0vNy40LjIzNjciLFsibG9hZE9yZGVyIl09OCxwcm9wVD17fS
↪  xbInN0YWNrRGVwdGgiXT0wLFsic3RhdHVzIl09ImFjdGl2ZS
↪  IsWyJ1c2VyTmFtZSJdPSJ2aW0iLH0seGFsdD17WyJmbiJdPS
↪  Ivc3cvc3VtbWl0L21vZHVsZWZpbGVzL3NpdGUvbGludXgtcm
↪  hlbDctcHBjNjRsZS9Db3JlL3hh";'
export '_ModuleTable005_;' '_ModuleTable006_="bHQvMS
↪  4xLjMubHVhIixbImZ1bGxOYW1lIl09InhhbHQvMS4xLjMiLF
↪  sibG9hZE9yZGVyIl09NCxwcm9wVD17fSxbInN0YWNrRGVwdG
↪  giXT0xLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZS
↪  JdPSJ4YWx0Iix9LHhsPXtbImZuIl09Ii9zdy9zdW1taXQvbW
↪  9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL0
↪  NvcmUveGwvMTYuMS4xLTEubHVhIixbImZ1bGxOYW1lIl09In
↪  hsLzE2LjEuMS0xIixbImxvYWRPcmRlciJdPTEscHJvcFQ9e3
↪  0sWyJzdGFja0RlcHRoIl09MSxbInN0YXR1cyJdPSJhY3Rpdm
↪  UiLFsidXNlck5hbWUiXT0ieGwiLH0sfSxtcGF0aEE9eyIvYX
↪  V0b2ZzL25jY3Mtc3ZtMV9zdy9zdW1taXQvbW9kdWxlZmlsZX
↪  Mvc2l0ZS9saW51eC1yaGVsNy1w";'
```

```
export '_ModuleTable006_;' '_ModuleTable007_="cGM2NG
↪ xlL3NwZWN0cnVtLW1waS8xMC4yLjAuMTEtMjAxOTAyMDEtNn
↪ F5cGQ2ci94bC8xNi4xLjEtMSIsIi9zdy9zdW1taXQvbW9kdW
↪ xlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL3hsLz
↪ E2LjEuMS0xIiwiL3N3L3N1bW1pdC9tb2R1bGVmaWxlcy9zaX
↪ RlL2xpbnV4LXJoZWw3LXBwYzY0bGUvQ29yZSIsIi9zdy9zdW
↪ 1taXQvbW9kdWxlZmlsZXMvY29yZSIsIi9zdy9zdW1taXQvbG
↪ 1vZC83LjcuMTAvcmhlbDcuM19nbnU0LjguNS9tb2R1bGVmaW
↪ xlcy9MaW51eCIsIi9zdy9zdW1taXQvbG1vZC83LjcuMTAvcm
↪ hlbDcuM19nbnU0LjguNS9tb2R1bGVmaWxlcy9Db3JlIiwiL3
↪ N3L3N1bW1pdC9sbW9kLzcuNy4xMC9yaGVsNy4zX2dudTQuOC
↪ 41L2xtb2QvbG1vZC9tb2R1bGVm";'
export '_ModuleTable007_;' '_ModuleTable008_="aWxlcy
↪ 9Db3JlIix9LFsic3lzdGVtQmFzZU1QQVRIIl09Ii9zdy9zdW
↪ 1taXQvbG1vZC83LjcuMTAvcmhlbDcuM19nbnU0LjguNS9tb2
↪ R1bGVmaWxlcy9MaW51eDovc3cvc3VtbWl0L2xtb2QvNy43Lj
↪ EwL3JoZWw3LjNfZ251NC44LjUvbW9kdWxlZmlsZXMvQ29yZT
↪ ovc3cvc3VtbWl0L2xtb2QvNy43LjEwL3JoZWw3LjNfZ251NC
↪ 44LjUvbG1vZC9sbW9kL21vZHVsZWZpbGVzL0NvcmUiLH0=";'
export '_ModuleTable008_;' '_ModuleTable_Sz_="8";'
export '_ModuleTable_Sz_;'
++ MODULEPATH=/autofs/nccs-svm1_sw/summit/modulefile
↪ s/site/linux-rhel7-ppc64le/spectrum-mpi/10.2.0.1
↪ 1-20190201-6qypd6r/xl/16.1.1-1:/sw/summit/module
↪ files/site/linux-rhel7-ppc64le/xl/16.1.1-1:/sw/s
↪ ummit/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
↪ /lmod/7.7.10/rhel7.3_gnu4.8.5/modulefiles/Core:/
↪ sw/summit/lmod/7.7.10/rhel7.3_gnu4.8.5/lmod/lmod
↪ /modulefiles/Core
++
export MODULEPATH
++ _ModuleTable001_=X01vZHVsZVRhYmxlXz17WyJNVHZlcnNp
↪ b24iXT0zLFsiY19yZWJ1aWxkVGltZSJdPWZhbHNlLFsiY19z
↪ aG9ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9e30sZmFtaWx5PXtb
↪ ImNvbXBpbGVyIl09InhsIixbIm1waSJdPSJzcGVjdHJ1bS1t
↪ cGkiLH0sbVQ9e0RlZkFwcHM9e1siZm4iXT0iL3N3L3N1bW1p
↪ dC9tb2R1bGVmaWxlcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0
↪ bGUvQ29yZS9EZWZBcHBzLmx1YSIsWyJmdWxsTmFtZSJdPSJE
↪ ZWZBcHBzIixbImxvYWRPcmRlciJdPTYscHJvcFQ9e30sWyJz
↪ dGFja0RlcHRoIl09MCxbInN0YXR1cyJdPSJhY3RpdmUiLFsi
↪ dXNlck5hbWUiXT0iRGVmQXBwcyIsfSxjdWRhPXtbImZuIl09
↪ Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1y
++
export _ModuleTable001_
++ _ModuleTable002_=aGVsNy1wcGM2NGxlL3hsLzE2LjEuMS0x
↪ L2N1ZGEvOS4yLjE0OC5sdWEiLFsiZnVsbE5hbWUiXT0iY3Vk
↪ YS85LjIuMTQ4IixbImxvYWRPcmRlciJdPTEwLHByb3BUPXt9
↪ LFsic3RhY2tEZXB0aCJdPTAsWyJzdGF0dXMiXT0iYWN0aXZl
↪ IixbInVzZXJOYW1lIl09ImN1ZGEiLH0sZmZ0dz17WyJmbiJd
↪ PSIvYXV0b2ZzL25jY3Mtc3ZtMV9zdy9zdW1taXQvbW9kdWxl
↪ ZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL3NwZWN0
↪ cnVtLW1waS8xMC4yLjAuMTEtMjAxOTAyMDEtNnF5cGQ2ci94
↪ bC8xNi4xLjEtMS9mZnR3LzMuMy44Lmx1YSIsWyJmdWxsTmFt
↪ ZSJdPSJmZnR3LzMuMy44IixbImxvYWRPcmRlciJdPTkscHJv
↪ cFQ9e30sWyJzdGFja0RlcHRoIl09MCxbInN0YXR1cyJdPSJh
++
export _ModuleTable002_
++ _ModuleTable003_=Y3RpdmUiLFsidXNlck5hbWUiXT0iZmZ0
↪ dyIsfSxoc2k9e1siZm4iXT0iL3N3L3N1bW1pdC9tb2R1bGVm
↪ aWxlcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0bGUvQ29yZS9o
↪ c2kvNS4wLjIucDUubHVhIixbImZ1bGxOYW1lIl09ImhzaS81
↪ LjAuMi5wNSIsWyJsb2FkT3JkZXIiXT0zLHByb3BUPXt9LFsi
↪ c3RhY2tEZXB0aCJdPTEsWyJzdGF0dXMiXT0iYWN0aXZlIixb
↪ InVzZXJOYW1lIl09ImhzaSIsfSxbImxzZi10b29scyJdPXtb
↪ ImZuIl09Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9s
↪ aW51eC1yaGVsNy1wcGM2NGxlL0NvcmUvbHNmLXRvb2xzLzIu
↪ MC5sdWEiLFsiZnVsbE5hbWUiXT0ibHNmLXRvb2xzLzIuMCIs
↪ WyJsb2FkT3JkZXIiXT01LHByb3BUPXt9LFsic3RhY2tEZXB0
++
export _ModuleTable003_
++ _ModuleTable004_=aCJdPTEsWyJzdGF0dXMiXT0iYWN0aXZl
↪ IixbInVzZXJOYW1lIl09ImxzZi10b29scyIsfSxbInNwZWN0
↪ cnVtLW1waSJdPXtbImZuIl09Ii9zdy9zdW1taXQvbW9kdWxl
↪ ZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1wcGM2NGxlL3hsLzE2
↪ LjEuMS0xL3NwZWN0cnVtLW1waS8xMC4yLjAuMTEtMjAxOTAy
↪ MDEubHVhIixbImZ1bGxOYW1lIl09InNwZWN0cnVtLW1waS8x
↪ MC4yLjAuMTEtMjAxOTAyMDEiLFsibG9hZE9yZGVyIl09Mixw
↪ cm9wVD17fSxbInN0YWNrRGVwdGgiXT0xLFsic3RhdHVzIl09
↪ ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJzcGVjdHJ1bS1tcGki
↪ LH0sdG11eD17WyJmbiJdPSIvc3cvc3VtbWl0L21vZHVsZWZp
↪ bGVzL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS9Db3JlL3Rt
++
export _ModuleTable004_
++ _ModuleTable005_=dXgvMi4yLmx1YSIsWyJmdWxsTmFtZSJd
↪ PSJ0bXV4LzIuMiIsWyJsb2FkT3JkZXIiXT03LHByb3BUPXt9
↪ LFsic3RhY2tEZXB0aCJdPTAsWyJzdGF0dXMiXT0iYWN0aXZl
↪ IixbInVzZXJOYW1lIl09InRtdXgiLH0sdmltPXtbImZuIl09
↪ Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1y
↪ aGVsNy1wcGM2NGxlL0NvcmUvdmltLzcuNC4yMzY3Lmx1YSIs
↪ WyJmdWxsTmFtZSJdPSJ2aW0vNy40LjIzNjciLFsibG9hZE9y
↪ ZGVyIl09OCxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0wLFsi
↪ c3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJ2aW0i
↪ LH0seGFsdD17WyJmbiJdPSIvc3cvc3VtbWl0L21vZHVsZWZp
↪ bGVzL3NpdGUvbGludXgtcmhlbDctcHBjNjRsZS9Db3JlL3hh
++
export _ModuleTable005_
++ _ModuleTable006_=bHQvMS4xLjMubHVhIixbImZ1bGxOYW1l
↪ Il09InhhbHQvMS4xLjMiLFsibG9hZE9yZGVyIl09NCxwcm9w
↪ VD17fSxbInN0YWNrRGVwdGgiXT0xLFsic3RhdHVzIl09ImFj
↪ dGl2ZSIsWyJ1c2VyTmFtZSJdPSJ4YWx0Iix9LHhsPXtbImZu
↪ Il09Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51
↪ eC1yaGVsNy1wcGM2NGxlL0NvcmUveGwvMTYuMS4xLTEubHVh
↪ IixbImZ1bGxOYW1lIl09InhsLzE2LjEuMS0xIixbImxvYWRP
↪ cmRlciJdPTEscHJvcFQ9e30sWyJzdGFja0RlcHRoIl09MSxb
↪ InN0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5hbWUiXT0ieGwi
↪ LH0sfSxtcGF0aEE9eyIvYXV0b2ZzL25jY3Mtc3ZtMV9zdy9z
↪ dW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1yaGVsNy1w
++
export _ModuleTable006_
```

```
++ _ModuleTable007_=cGM2NGxlL3NwZWN0cnVtLW1waS8xMC4y
↪  LjAuMTEtMjAxOTAyMDEtNnF5cGQ2ci94bC8xNi4xLjEtMSIs
↪  Ii9zdy9zdW1taXQvbW9kdWxlZmlsZXMvc2l0ZS9saW51eC1y
↪  aGVsNy1wcGM2NGxlL3hsLzE2LjEuMS0xIiwiL3N3L3N1bW1p
↪  dC9tb2R1bGVmaWxlcy9zaXRlL2xpbnV4LXJoZWw3LXBwYzY0
↪  bGUvQ29yZSIsIi9zdy9zdW1taXQvbW9kdWxlZmlsZXMvY29y
↪  ZSIsIi9zdy9zdW1taXQvbG1vZC83LjcuMTAvcmhlbDcuM19n
↪  bnU0LjguNS9tb2R1bGVmaWxlcy9MaW51eCIsIi9zdy9zdW1t
↪  aXQvbG1vZC83LjcuMTAvcmhlbDcuM19nbnU0LjguNS9tb2R1
↪  bGVmaWxlcy9Db3JlIiwiL3N3L3N1bW1pdC9sbW9kLzcuNy4x
↪  MC9yaGVsNy4zX2dudTQuOC41L2xtb2QvbG1vZC9tb2R1bGVm
++
export _ModuleTable007_
++ _ModuleTable008_=aWxlcy9Db3JlIix9LFsic3lzdGVtQmFz
↪  ZU1QQVRIIl09Ii9zdy9zdW1taXQvbG1vZC83LjcuMTAvcmhl
↪  bDcuM19nbnU0LjguNS9tb2R1bGVmaWxlcy9MaW51eDovc3cv
↪  c3VtbWl0L2xtb2QvNy43LjEwL3JoZWw3LjNfZ251NC44LjUv
↪  bW9kdWxlZmlsZXMvQ29yZTovc3cvc3VtbWl0L2xtb2QvNy43
↪  LjEwL3JoZWw3LjNfZ251NC44LjUvbG1vZC9sbW9kL21vZHVs
↪  ZWZpbGVzL0NvcmUiLH0=
++
export _ModuleTable008_
++ _ModuleTable_Sz_=8
++
export _ModuleTable_Sz_
++ : -s sh
+ eval
+ nvidia-smi
Wed Apr 10 13:27:42 2019
+-----------------------------------------------------
↪  -------------------------+
| NVIDIA-SMI 396.64                 Driver Version:
↪  396.64                     |
|-------------------------------+----------------------
↪  ---+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A
↪  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage
↪  | GPU-Util  Compute M. |
|===============================+======================
↪  ===+======================|
|   0  Tesla V100-SXM2...  On   | 00000004:04:00.0 Off
↪  |                    0 |
| N/A   38C    P0    37W / 300W |  14432MiB / 16128MiB
↪  |      0%   E. Process |
+-------------------------------+----------------------
↪  ---+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000004:05:00.0 Off
↪  |                    0 |
| N/A   45C    P0    38W / 300W |  14684MiB / 16128MiB
↪  |      0%   E. Process |
+-------------------------------+----------------------
↪  ---+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000035:03:00.0 Off
↪  |                    4 |
| N/A   38C    P0    38W / 300W |  14458MiB / 16128MiB
↪  |      0%   E. Process |
+-------------------------------+----------------------
↪  ---+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000035:04:00.0 Off
↪  |                    0 |
| N/A   48C    P0    41W / 300W |  15451MiB / 16128MiB
↪  |      0%   E. Process |
+-------------------------------+----------------------
↪  ---+----------------------+

+-----------------------------------------------------
↪  -------------------------+
| Processes:
↪                GPU Memory |
|  GPU       PID   Type   Process name
↪                Usage      |
|=====================================================
↪  =========================|
|  No running processes found
↪                           |
+-----------------------------------------------------
↪  -------------------------+
+ lshw -short -quiet -sanitize
+ cat
/ccs/home/kiran92/scripts/SC19AD-script.sh: line 19:
↪  lshw: command not found
+ lspci
0000:00:00.0 PCI bridge: IBM Device 04c1
0000:01:00.0 Non-Volatile memory controller: Samsung
↪  Electronics Co Ltd NVMe SSD Controller 172Xa (rev
↪  01)
0001:00:00.0 PCI bridge: IBM Device 04c1
0001:01:00.0 USB controller: Texas Instruments
↪  TUSB73x0 SuperSpeed USB 3.0 xHCI Host Controller
↪  (rev 02)
0002:00:00.0 PCI bridge: IBM Device 04c1
0002:01:00.0 PCI bridge: ASPEED Technology, Inc.
↪  AST1150 PCI-to-PCI Bridge (rev 04)
0002:02:00.0 VGA compatible controller: ASPEED
↪  Technology, Inc. ASPEED Graphics Family (rev 41)
0003:00:00.0 PCI bridge: IBM Device 04c1
0003:01:00.0 Infiniband controller: Mellanox
↪  Technologies MT28800 Family [ConnectX-5 Ex]
0003:01:00.1 Infiniband controller: Mellanox
↪  Technologies MT28800 Family [ConnectX-5 Ex]
0004:00:00.0 PCI bridge: IBM Device 04c1
0004:01:00.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0004:01:00.1 System peripheral: PLX Technology, Inc.
↪  Device 87d0 (rev ca)
0004:01:00.2 System peripheral: PLX Technology, Inc.
↪  Device 87d0 (rev ca)
0004:01:00.3 System peripheral: PLX Technology, Inc.
↪  Device 87d0 (rev ca)
```

```
0004:01:00.4 System peripheral: PLX Technology, Inc.
↪  Device 87d0 (rev ca)
0004:02:02.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0004:02:0a.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0004:02:0b.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0004:02:0c.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0004:03:00.0 SATA controller: Marvell Technology
↪  Group Ltd. 88SE9235 PCIe 2.0 x2 4-port SATA 6 Gb/s
↪  Controller (rev 11)
0004:04:00.0 3D controller: NVIDIA Corporation
↪  GV100GL [Tesla V100 SXM2] (rev a1)
0004:05:00.0 3D controller: NVIDIA Corporation
↪  GV100GL [Tesla V100 SXM2] (rev a1)
0005:00:00.0 PCI bridge: IBM Device 04c1
0005:01:00.0 Ethernet controller: Broadcom Limited
↪  NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
0005:01:00.1 Ethernet controller: Broadcom Limited
↪  NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
0006:00:00.0 Bridge: IBM Device 04ea (rev 01)
0006:00:00.1 Bridge: IBM Device 04ea (rev 01)
0006:00:00.2 Bridge: IBM Device 04ea (rev 01)
0006:00:01.0 Bridge: IBM Device 04ea (rev 01)
0006:00:01.1 Bridge: IBM Device 04ea (rev 01)
0006:00:01.2 Bridge: IBM Device 04ea (rev 01)
0007:00:00.0 Bridge: IBM Device 04ea (rev 01)
0007:00:00.1 Bridge: IBM Device 04ea (rev 01)
0007:00:00.2 Bridge: IBM Device 04ea (rev 01)
0007:00:01.0 Bridge: IBM Device 04ea (rev 01)
0007:00:01.1 Bridge: IBM Device 04ea (rev 01)
0007:00:01.2 Bridge: IBM Device 04ea (rev 01)
0030:00:00.0 PCI bridge: IBM Device 04c1
0030:01:00.0 Ethernet controller: Broadcom Limited
↪  NetXtreme II BCM57800 1/10 Gigabit Ethernet (rev
↪  10)
0030:01:00.1 Ethernet controller: Broadcom Limited
↪  NetXtreme II BCM57800 1/10 Gigabit Ethernet (rev
↪  10)
0030:01:00.2 Ethernet controller: Broadcom Limited
↪  NetXtreme II BCM57800 1/10 Gigabit Ethernet (rev
↪  10)
0030:01:00.3 Ethernet controller: Broadcom Limited
↪  NetXtreme II BCM57800 1/10 Gigabit Ethernet (rev
↪  10)
0033:00:00.0 PCI bridge: IBM Device 04c1
0033:01:00.0 Infiniband controller: Mellanox
↪  Technologies MT28800 Family [ConnectX-5 Ex]
0033:01:00.1 Infiniband controller: Mellanox
↪  Technologies MT28800 Family [ConnectX-5 Ex]
0034:00:00.0 PCI bridge: IBM Device 04c1
0035:00:00.0 PCI bridge: IBM Device 04c1
0035:01:00.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0035:02:04.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0035:02:05.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0035:02:0d.0 PCI bridge: PLX Technology, Inc. Device
↪  8725 (rev ca)
0035:03:00.0 3D controller: NVIDIA Corporation
↪  GV100GL [Tesla V100 SXM2] (rev a1)
0035:04:00.0 3D controller: NVIDIA Corporation
↪  GV100GL [Tesla V100 SXM2] (rev a1)
```