Scaling Single-Image Super-Resolution Training on Modern HPC Clusters: Early Experiences

Quentin Anthony, Lang Xu, Hari Subramoni, and Dhabaleswar K. (DK) Panda
Dept. of Computer Science and Engineering
The Ohio State University
{anthony.301, xu.3304, subramoni.1, panda.2}@osu.edu

***Abstract*—Deep Learning (DL) models for super-resolution (DLSR) are an emerging trend in response to the growth of ML/DL applications requiring high-resolution images. DLSR methods have also shown promise in domains such as medical imaging, surveillance, and microscopy. However, DLSR models are extremely computationally demanding, and require unreasonably long training times on modern Volta GPUs. In our experiments, we observed only 10.3 images/second on a single Volta GPU for training EDSR, a state-of-the-art DLSR model for single-image super-resolution. In comparison, a Volta GPU can process 360 images/second while training ResNet-50, a state-of-the-art model for image classification. Therefore, we believe supercomputers provide a good candidate to speed up DLSR model training. In this paper, we select EDSR as the representative DLSR PyTorch model. Further, we introduce Horovod-based distributed EDSR training. However, we observed poor default EDSR scaling performance on the Lassen HPC system at Lawrence Livermore National Laboratory. To investigate the performance degradations, we perform exhaustive communication profiling. These profiling insights are then used to optimize CUDA-Aware MPI for DLSR models by ensuring advanced MPI designs involving CUDA IPC and registration caching are properly applied by DL frameworks. We present a comprehensive scaling study of EDSR with MVAPICH2-GDR and NCCL up to 512 GPUs on Lassen. We demonstrate an improvement in scaling efficiency by 15.6% over default Horovod training, which translates to a $1.26\times$ speedup in training performance.**

***Index Terms*—DNN Training, Performance Characterization, MVAPICH2 MPI, PyTorch, Horovod, Super Resolution**

I. Introduction and Motivation

Deep Neural Networks (DNNs)[1] have seen widespread adoption in many computer vision tasks, and are pioneering state-of-the-art image processing problems such as image classification, image segmentation, and image super-resolution. A DNN is a statistical model that trains a set of parameters to learn unspecified relationships in data. An input *x* is passed through the network and non-linearly mapped to a learned output *y*. We label nodes in the graph as *neurons*, and a set of neurons grouped at the same level as *layers*. When processing image data, each image is passed through each layer of the DNN sequentially. After each forward pass through the DNN, a loss value will be computed according to the difference between the model's output and the expected output. To reduce the loss value, the neuron parameters will be adjusted to produce a function f such that $y = f(x)$. These parameter adjustments are performed during the backward pass when the loss gradients are propagated backwards through the DNN. After a large number of forward/backward passes (iterations), the trained DNN can be used to output predictions given a new set of images. **Training DNN models requires significant computation and communication resources, making HPC systems an ideal environment to conduct relevant research [1] [2] [3] [4].**

[1]We use the terms DNN and model interchangeably in this paper

Recently, deep neural networks provide significantly improved performance in terms of peak signal-to-noise ratio (PSNR) in image super-resolution problems [5]. A standard deep learning super-resolution (DLSR) model is SRResNet in which the structure of the network mostly mimics the ResNet architecture from He et al [6]. However, the ResNet architecture is meant for high-level computer vision tasks such as image classification and object detection. As a result, architectures such as ResNet architecture are typically suboptimal for low-level vision problems like image super-resolution [5].

A better DCNN for such low-level vision tasks should have fewer modules than high-level networks and should be modified to a configuration that will be best for image super-resolution. EDSR is one successful example of a DLSR model. A key difference between conventional image classification models (e.g. ResNet-50) and a super-resolution model (e.g. EDSR) is that super-resolution models are significantly more intensive on memory footprint and computation cost (see Figure 1). From the previous exploration on computer vision tasks performed in [7] and given that there exists few studies that feature feasibility and performance of DLSR training on HPC systems, studying the EDSR training process on HPC systems is a timely contribution to the DL and HPC community.

To the best of our knowledge, only the work of Zhang [8] has scaled an image super-resolution model to an HPC system on 16 GPUs using synchronized data parallelism with Horovod. We demonstrated that comparable scaling performance can be achieved across a much larger number of nodes (512 GPUs) by optimizing the MPI layer to enable CUDA inter-process communication (IPC) and registration cache designs for DLSR workloads. In this work, we first modify the

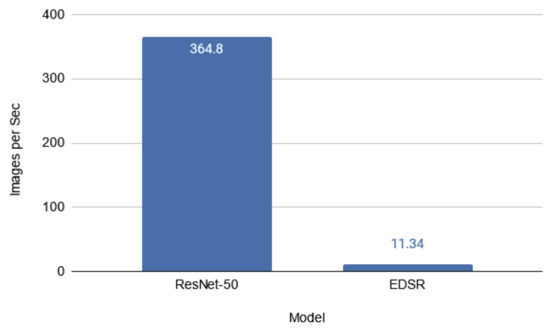


Fig. 1: Single-node performance for classification (ResNet-50) and super-resolution (EDSR) models on a V100 GPU

existing SR network (EDSR) written in PyTorch to provide multi-node data-parallel support via Horovod, then we examined the default scaling behavior on MVAPICH2-GDR and NCCL communication backends. Finally, we demonstrated the potential performance benefits that can be gained by ensuring MPI-level enhancements are properly used by the target DL framework.

A. Challenges

The key challenge addressed in this paper is: *How do we efficiently distribute DLSR models after taking into account the unique characteristics of both the DLSR model and the HPC system?* We seek distributed training optimizations that may be easily applied to other novel DNN architectures. To answer this broad question, we solve the following concrete challenges:

- What are the key communication properties of DLSR networks that affect their training performance at scale?
- What MPI-level optimizations can be made to efficiently train DLSR models on distributed systems, given these key differences?
- What benefits can established MPI-level enhancements (e.g. CUDA IPC and InfiniBand registration caching) provide to emerging DLSR models?

B. Proposed Approach

To solve these challenges, we used a three-step process to distribute existing DLSR models. First, introduce data-parallel training support via Horovod to the existing DLSR training code. Second, exhaustively profile the MPI layer of the stack to gain insight into the communication properties of the model. Third, resolve performance bottlenecks uncovered in step two using optimizations at the MPI layer.

C. Contributions

This work provides a mapping from MPI-level enhancements to DLSR training performance improvements. Further, our proposed training approach is agnostic to the model, DL framework, and system used for DNN training. To understand the communication characteristics, we rely on a diagnostic tool called *hvprof* [9], which is agnostic to the DL framework, communication backend, and system. We make the following key contributions in this paper:

- Demonstrate the productivity and performance shortcomings of existing distributed DLSR training methods.
- Establish the usefulness of Horovod and MPI profiling tools for distributing novel DNNs on GPU clusters, and apply them to demonstrate a 45.4% improvement (Table I) in total MPI_Allreduce time.
- Ensure MPI-level optimizations are properly used by the DL framework, and measure their affects on training throughput.
- We demonstrate the superiority of our optimized Horovod/MPI approach and report an improvement in scaling efficiency by 15.6% (Fig. 13), which translates to a $1.26\times$ speedup over the default approach.

D. Organization

The rest of the paper is organized as follows. Section II provides the necessary background, including details on Horovod, DLSR, and CUDA-Aware MPI. Section III contains an overview of the proposed optimizations to improve distributed training performance. Section IV discusses characterization metrics, software libraries, and platforms used in this study. Sections V and VI provide insights into hyperparameter optimization for single-node training and the scaling trend under Horovod and MVAPICH2-GDR's default settings, respectively. In Section VII, we show optimized scaling performance using our proposed design, and then we summarize key insights in Section VIII. Section IX contains related work, and we conclude with Section X.

II. Background

A. CUDA Inter-Process Communication

CUDA Inter-Process Communication (IPC) is a well-established (since CUDA 4.1) interface that supports efficient inter-process communication on GPU device memory. This enables a process to share its GPU device buffer to a remote process located on the same node. Then, the remote process can map this device buffer into its own address space before issuing operations like cuMemcpy directly to it.

Formally, a GPU-to-GPU transfer via CUDA IPC is set up with the following steps:

1) A process creates an Inter-Process Communication (IPC) handle on its device buffer by using cuIpcGetMemHandle.
2) It then sends this handle to the remote process through host-based communication.
3) The remote process calls cuIpcOpenMemHandle on this handle to map the associated buffer into its local address space.

After the completion of the above steps, an MPI library may now use the CUDA IPC handle to efficiently transfer data within the GPUs on each node of the process group.

B. DL Frameworks

DL frameworks enable a transparent way for users to define, train, and validate on various CPU and GPU architectures. By assembling modular building blocks provided by DL frameworks, novel and reproducible DNN structures are able to be produced at minimal development cost. These frameworks enable users to define and implement models targeting specific end applications by hiding low-level mathematics and algorithms. The DNN architecture, the size of dataset and the DL framework in use are the three key components affecting DNN training time. At an age where DNN training is requiring increasing computation intensity, DL frameworks that include accessible distributed training features are scarce. In addition to limited distributed support in DL frameworks, we also need to take various communication backends into consideration for the distributed support that does exist. As an example, a number of distributed TensorFlow backends exist including gRPC, gRPC-X, and MPI/NCCL. [10].

C. Data-Parallelism

Data parallelism is a popular approach to distributed DNN training that first duplicates the DNN model to each CPU/GPU before partitioning the training data across all CPUs/GPUs. The number of data samples sent to each CPU/GPU at each global training step is known as the *batch size*. After each training step, DNN parameters need to be synchronized by averaging the gradients among all processes. This is typically achieved with an MPI_allreduce, which performs an element-wise sum operation and sends the result to every process. The standard data-parallelism approach, synchronous training, requires each device's model copy to send an updated gradient before progressing to the next global training step. While asynchronous training approaches can improve the throughput on individual nodes, training convergence becomes complicated to achieve.

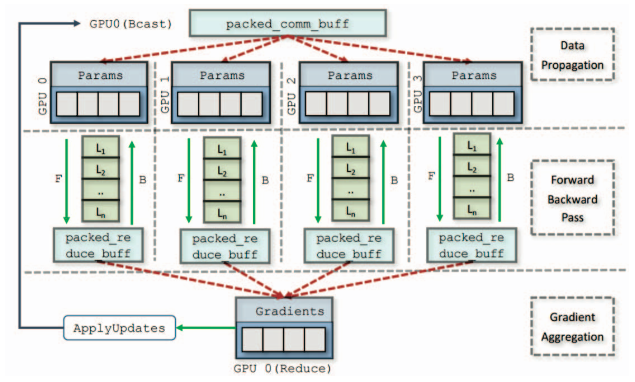


Fig. 2: Example data parallel DNN training on 4 GPUs [11]

D. Horovod

Horovod is a distributed DL framework that employs data parallelism to train DNNs [12]. Horovod primarily uses MPI_Allreduce and MPI_Bcast to implement data-parallel training. Horovod supports most communication backends including MPI, DDL [13], and NCCL [14]. Horovod handles synchronization across processes with a communication engine and optimizes distributed training performance by applying techniques such as Tensor Fusion. Tensor Fusion is an optimization technique that groups small allreduce operations into a single reduction operation that enhance DNN training performance. Tensor Fusion works as follows:

1) Determine which tensors are ready to be reduced. Select first few tensors that fit in `HOROVOD_FUSION_THRESHOLD` bytes and have the same data type.
2) Allocate fusion buffer of size. `HOROVOD_FUSION_THRESHOLD` if it was not allocated before. Default fusion buffer size is 64 MB.
3) Copy data of selected tensors into the fusion buffer.
4) Execute the allreduce operation on the fusion buffer.
5) Copy data from the fusion buffer into the output tensors.
6) repeat until there are no more update tensors to reduce in current cycle of length HOROVOD_CYCLE_TIME (Default is 3.5 ms).

For all evaluations in this paper, the HOROVOD_FUSION_THRESHOLD and HOROVOD_CYCLE_TIME are carefully tuned at each scale to maximize training throughput according to [7]. Currently, Horovod supports the TensorFlow, MXNet, PyTorch, and Keras DL frameworks. Horovod acts as a middleware between these frameworks and a communication backend as depicted in Figure 3.

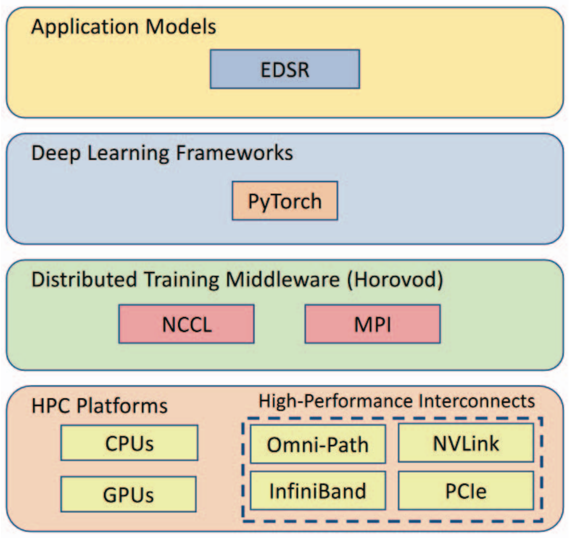


Fig. 3: Overview of a Distributed DNN Training Stack

E. Single-Image Super-Resolution

In single image super-resolution(SISR), researchers focus on finding the corresponding high resolution (HR) image from the low resolution (LR) image. SISR is applicable to a wide range of relevant applications such as medical and satellite imaging, image restoration, and microscopy. In general, the LR training images can be obtained by downsampling HR target images. Since the degradation process is unknown and can be changed by a variety of factors including compression artifacts, anisotropic degradations, sensor noise, speckle noise, etc, there always exist multiple SR output images in accordance to one input LR image. Several classical SR methods were introduced such as prediction-based, edge-based, statistical, patch-based, and sparse representation techniques, etc. [15]

Given the emergence of DL techniques in recent years, DL super-resolution (DLSR) models began to gain attention. Examples range from the early Convolutional Neural Network (CNN)-based method (e.g. SRCNN) to Generative Adversarial Nets (GAN) (e.g. SRGAN). DLSR models possess a wide range of DNN architectures, loss functions, learning strategies, and principles [15]. For an example output comparing traditional methods and EDSR, see Figure 4.

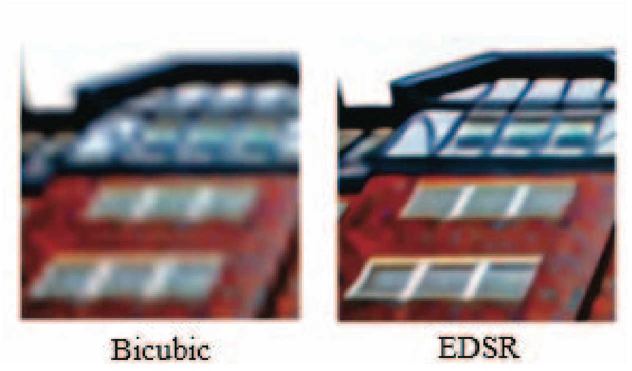
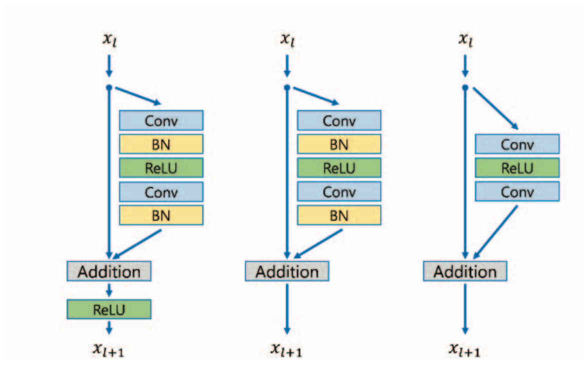
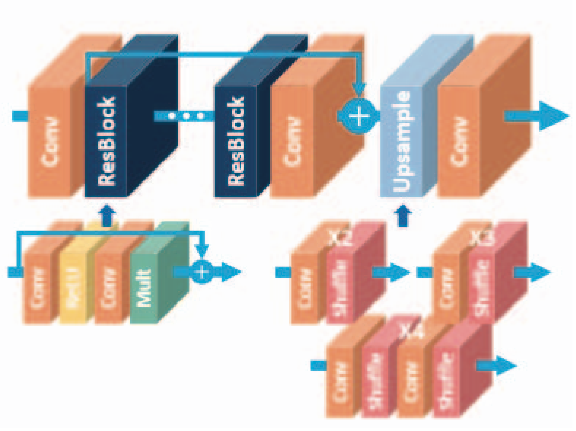


Fig. 4: Examples of HR images output using traditional bicubic upsampling and EDSR [5]

In order to provide application-agnostic benchmarks for image SR models, a variety of datasets have been made available. some of those provide LR-HR image pairs while others may only provide HR images. Common ones include Set5, Set14, Urban100 and DIV2K. [15] DIV2K is a novel image dataset that features diverse 2K resolution high-quality images collected from the Internet. It includes 1000 images at a higher resolution than most other popular datasets. [16] The 1000 images consist of 800 train, 100 validation and 100 test images. We used DIV2K for EDSR evaluation because of its diversity and size. The accuracy and performance of a SR model can be determined using several Image Quality Assessment (IQA) methods such as Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) etc. [15] [17]



(a) Comparison between residual blocks in original ResNet, SRResNet and EDSR (left to right) [5].



(b) General architecture of EDSR [5].

Fig. 5: EDSR Architecture improvements.

F. EDSR

Enhanced Deep Super-Resolution network (EDSR) serves as an improved version of traditional residual learning techniques. EDSR is one of the most popular image SR models and addressed DNN limitations in architecture optimality and performance shortage by analyzing and removing unnecessary modules to simplify the network architecture. Further, EDSR makes use of an appropriate loss function and makes careful model modifications while training. An illustration of the EDSR architecture is included in Figure 5. For a comparison of residual blocks among different models, please refer to Figure 5a. EDSR was evaluated using the standard benchmark datasets along with the DIV2K dataset. The proposed EDSR network outperformed all other solutions regarding PSNR and SSIM, and ranked first in the NTIRE 2017 Super-Resolution Challenge [5] [18]. We chose EDSR as one of our evaluation model due to its popularity, performance, and open-source implementation.

III. Proposed Optimization Approach

We now describe our proposed approach to improve the performance of EDSR distributed training. Broadly, we follow a three-phase optimization approach:

1) As EDSR's default implementation is only for a single CPU/GPU, we must first realize a distributed version of EDSR. Based on the findings in [10], we choose Horovod to implement distributed EDSR. *Design details are discussed further in Section III-A.*

2) With a basic version of distributed EDSR in hand, we identify performance bottlenecks using an in-house Horovod profiler called *hvprof* [9]. The major insight is that the default implementation using Horovod is not efficient at scale (See Figure 13). *More details are described in Section III-B.*

3) Given the identified bottlenecks, we enhance MVAPICH2-GDR to provide optimized distributed training with Horovod and PyTorch for EDSR workloads. *More details are described in Sections III-C and III-D*

A. Extending EDSR to Support Horovod

Given the largely formulaic structure of DNN training using a DL framework, Horovod support can be added in a model-agnostic manner. DNN training with a DL framework broadly follows the following guidelines:

1) Setup training data and apply preprocessing, if necessary.
2) Define model structure.
3) Declare optimizer and training hyperparameters.
4) Create a training loop or computational graph to carry out each training iteration.

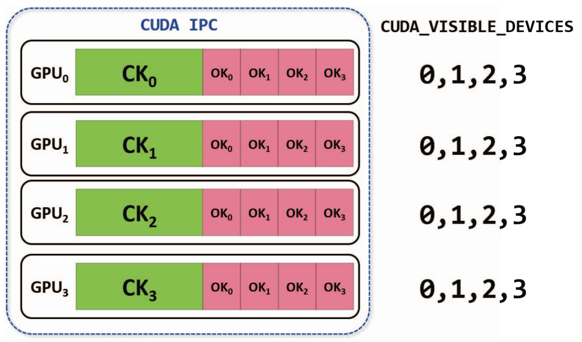
With this DNN training pipeline in mind, we added Horovod support to EDSR by following the practice described by Horovod developers. The guidelines are as follows.

1) Map the processes to the GPUs on each node (typically one GPU per process).
2) Add a Horovod broadcast operation to set up the initial model parameters at each device.
3) Wrap the training optimizer in Horovod's distributed optimizer.
4) Scale the learning rate of the optimizer by the number of devices (Optional, but good practice to counteract the effective increased batch size).
5) Add logging at each training step to monitor training.

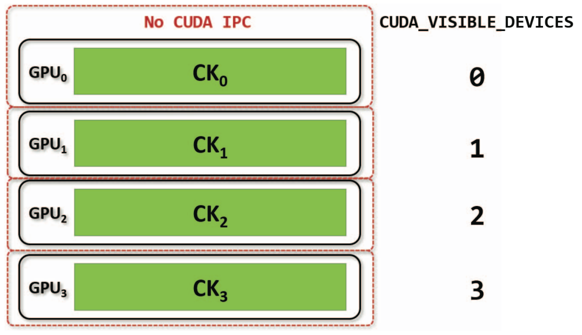
Following this approach, we have added Horovod distributed training to an implementation of EDSR in PyTorch.

B. Profiling and Improving Performance of Communication

Our goal is to improve performance while maintaining a model-agnostic approach. MVAPICH2-GDR [19], a GPU-Direct RDMA (GDR) MPI implementation, can provide competitive performance without sacrificing user productivity or making explicit changes to the application code. We collected the default distributed DNN training performance data discussed in Section VI.


CUDA IPC
CUDA_VISIBLE_DEVICES
GPU$_0$ CK$_0$ OK$_0$ OK$_1$ OK$_2$ OK$_3$ 0,1,2,3
GPU$_1$ CK$_1$ OK$_0$ OK$_1$ OK$_2$ OK$_3$ 0,1,2,3
GPU$_2$ CK$_2$ OK$_0$ OK$_1$ OK$_2$ OK$_3$ 0,1,2,3
GPU$_3$ CK$_3$ OK$_0$ OK$_1$ OK$_2$ OK$_3$ 0,1,2,3

(a) Example of a DL framework's faulty mapping behavior. Each process launches a small overhead kernel (OK) on each device



(b) Previous method for avoiding unnecessary GPU cross-talk

Fig. 6: GPU mapping shortcomings in Python libraries

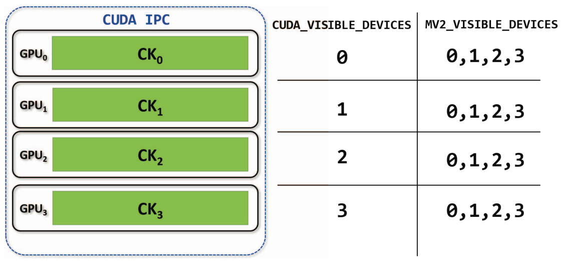
We ran an EDSR training job for 100 steps on 4 GPUs with *hvprof* [9], which provides the user with a detailed profile of Horovod's communication backend performance (MPI or NCCL) organized by message size and collective used. The *hvprof* output is depicted in Figure 14 and a speedup achievable for allreduce is in Section VII. After applying *hvprof* to a training run, we discovered the following key insight into the communication bottlenecks present in the basic version of DLSR models.

- Large messages are being sent inefficiently in MVAPICH2-GDR because DL frameworks are in conflict with CUDA IPC.

C. Restoring CUDA IPC to MPI for DL Frameworks

Many of the efficient large-message designs for pt-to-pt [20] and MPI_Allreduce [21] in MVAPICH2-GDR are heavily reliant on CUDA IPC. However, many Python libraries do not respect the GPU mappings defined in the user program. These libraries aggressively allocate GPU memory on all available devices, which can interfere with MPI execution and overflow GPU memory. In the best case, small sections of GPU memory are duplicated on each GPU by each process (see Figure 6a). These small "overhead kernels" (OK) leave significantly less space for the primary "compute kernel" (CK) that performs the forward/backward pass of each DNN training iteration. These extra kernels frequently overflow GPU memory, and restrict the hyperparameter space. This behavior is also present in DL frameworks such as PyTorch and distributed DL frameworks such as Horovod.

To combat this behavior, many Python libraries recommend that the user map the CUDA_VISIBLE_DEVICES environment variable to MPI's local ranks, which restricts which devices are visible to Python (and MPI) at the CUDA level. While this restricts Python libraries from running unnecessary kernels on remote GPUs, it also disables calls to CUDA IPC within MPI (see Figure 6), which relies on CUDA to share which devices within the node are available for IPC transfers. Therefore, MPI must default to main memory for all GPU transfers, which introduces significant overhead.



CUDA_VISIBLE_DEVICES	MV2_VISIBLE_DEVICES
0	0,1,2,3
1	0,1,2,3
2	0,1,2,3
3	0,1,2,3

Fig. 7: Proposed variable MV2_VISIBLE_DEVICES to restrict DL frameworks while maintaining CUDA IPC for MPI

As of CUDA 10.1, CUDA IPC no longer requires CUDA_VISIBLE_DEVICES to include multiple local GPUs for an IPC transfer to occur between them. Therefore, NCCL and CUDA-Aware MPI libraries are able to perform IPC transfers while the Python library is restricted to running kernels and allocating memory on its own local GPU. To implement this behavior, we introduce a flexible environment variable MV2_VISIBLE_DEVICES, which can be set to any combination of local GPUs on the system. MVAPICH2-GDR will be able to see these devices while CUDA_VISIBLE_DEVICES restricts the user application.

D. Registration Cache Optimization for PyTorch

MVAPICH2-GDR implements a registration cache which enables the zero-copy transfer of large messages across InfiniBand without memory registration if the communication buffer is used again and is within the cache [22]. Despite the benefits of registration caching across nodes (especially for large messages), previous versions of MVAPICH2-GDR have disabled the registration cache due to the conflicting custom memory allocators used in TensorFlow. PyTorch, however, does not require custom memory allocators. While it is not the primary focus of this work, we also explored the effect of MVAPICH2-GDR's registration cache designs for PyTorch.

For the remainder of this work, we refer to the optimized scenario in Figure 7 (and with the registration cache turned on) as **MPI-Opt**. Further, we refer to the previous method without CUDA IPC optimizations as **MPI**, and the same method with registration cache turned on as **MPI-Reg**. These results are discussed in detail in Sections VI and VII.

IV. Characterization Metrics and Platforms

We discuss different software libraries, evaluation platforms, and experiments needed to fully characterize EDSR training performance.

A. Evaluation Platforms

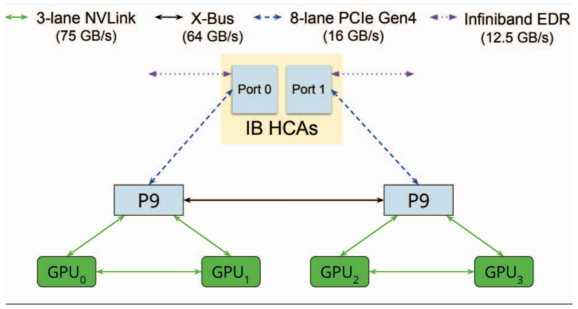


Fig. 8: Lassen Architecture

We performed all experiments on the Longhorn supercomputer at Texas Advanced Computing Center (TACC) and on the Lassen supercomputer at Lawerance Livermore National Laboratory (LLNL). Longhorn includes 96 nodes each with four NVIDIA Tesla V100 GPUs connected to a IBM Power 9 CPU via NVIDIA NVLink. Each Tesla V100 GPU has 16GB memory. Lassen is the #10-ranked machine in the TOP500 as of June 2019 [23] and is comprised of a total of 792 GPU nodes each with four NVIDIA Volta V100 GPU. Each node has a total of 44 Cores on IBM Power 9 architecture CPU. Each Volta V100 GPU has 16GB memory. For a look into Lassen's architecture, please refer to Figure 8.

B. Software Libraries

We use PyTorch v1.8.0 compiled with CUDA 10.2 and CUDNN 7.6.5 on Horovod 0.19.1. Horovod was built against the MVAPICH-GDR 2.3.5 GPU-direct MPI library [24]. Evaluations with NCCL used NCCL 2.8.3 from GitHub[2]. The EDSR model was pulled and modified from the publicly-available implementation[3]

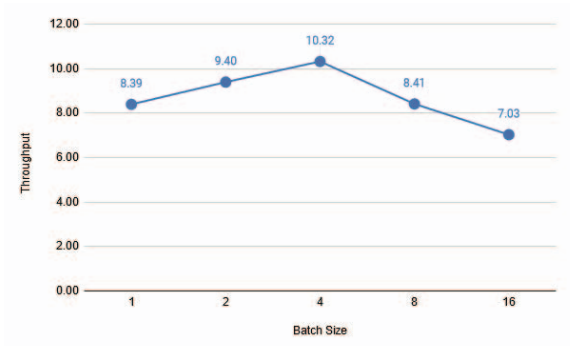


Fig. 9: Single-GPU Batch Size Evaluation

[2]https://github.com/NVIDIA/nccl
[3]https://github.com/thstkdgus35/EDSR-PyTorch

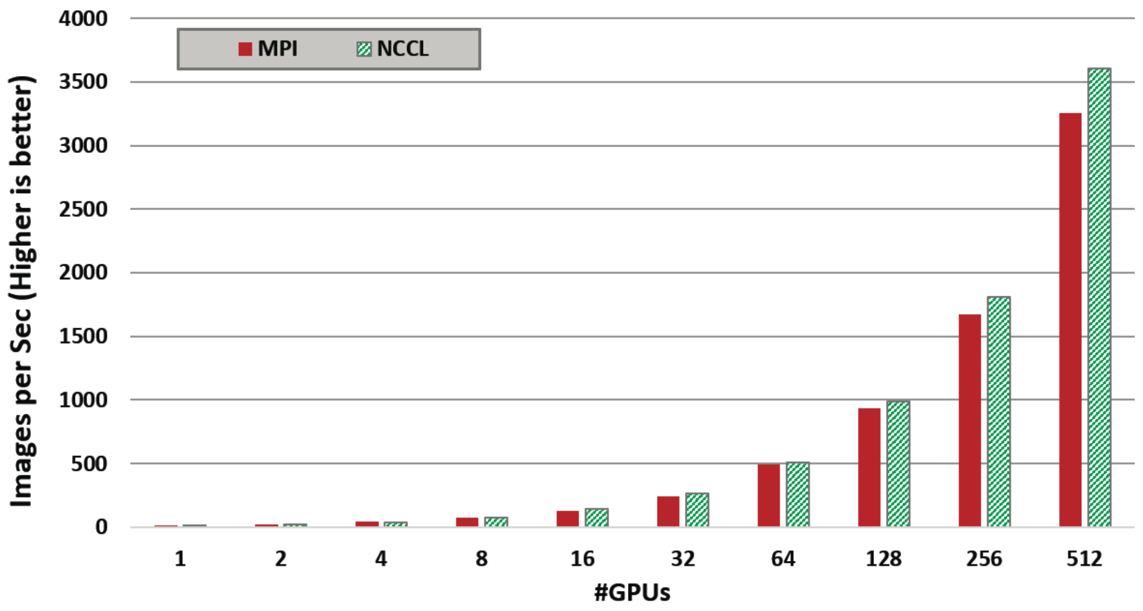


Fig. 10: Default Distributed EDSR Training Performance for Horovod built against MVAPICH2-GDR

C. EDSR Training

We used the standard EDSR training with a model of 32 residual blocks and 64 feature maps. We trained with an upscaling factor of 2 and a residual scaling of 0.1. After performing the single-node evaluation depicted in Figure 9, we chose a training batch size of 4 images.

V. Single-Node - Hyperparameter Optimization

Before distributing EDSR, we comprehensively profiled single-GPU and single-node training to find the best hyper-parameters to maximize throughput. In particular, we sought the best batch size and image patch size to maximize GPU utilization while maintaining the speed of convergence. For the single-node evaluation, we chose a batch-size of 4 and hyperparameters listed in Section IV-C.

VI. Default Scaling - Shortcomings and Insights

After default Horovod support was added to EDSR as detailed in Section III-A, we scale the training up to 128 Lassen nodes (512 V100 GPUs) by adding benchmarking support ($images/second$) to the EDSR model. Results between NCCL and MVAPICH2-GDR are compared to explore scaling behavior of each communication backend.

From Figures 10 and 13, we can see that, while performance is acceptable for a small number of nodes, throughput quickly degrades at scale. This is due to the conflict between the DL framework and CUDA IPC as detailed in Section III-C. Further, scaling efficiency drops below 60% for large node counts.

We then conducted an investigation into the system workload that SR models require to analyze the shortcomings of default scaling performance. We performed an analysis with our Horovod/MPI profiler *hvprof* as detailed in Section III-B.

VII. Optimized Scaling - Performance and Insights

Before investigating CUDA IPC for DL workloads, we investigated the performance impact of the MVAPICH2-GDR registration cache on EDSR training throughput scaling up to 128 Lassen nodes (512 GPUs). The results are depicted in Figure 11, and show an average improvement of **5.1%** in training throughput. Further, cache hit profiling data from these runs indicated an average cache hit rate of **93%**.

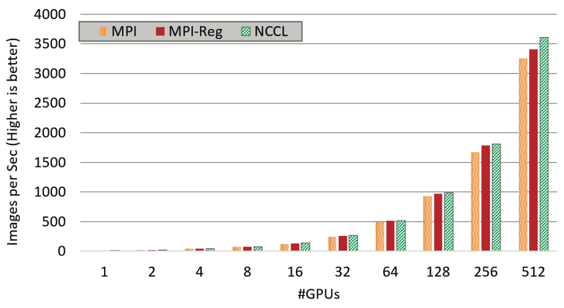


Fig. 11: Optimized EDSR Training Performance with registration cache for Horovod built against MVAPICH2-GDR

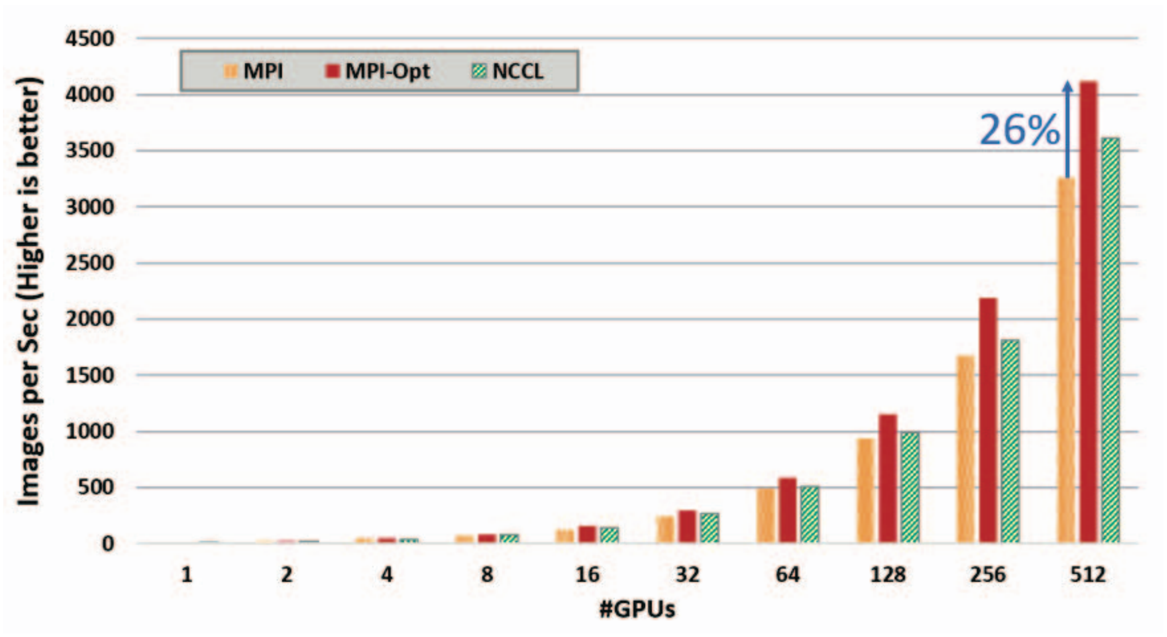


Fig. 12: Optimized Distributed EDSR Training Performance for Horovod built against MVAPICH2-GDR

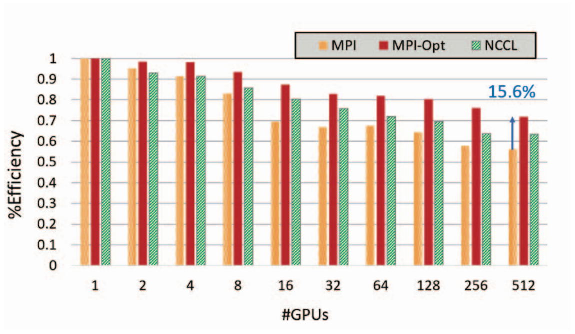


Fig. 13: Optimized EDSR Scaling Efficiency for Horovod built against MVAPICH2-GDR

After applying the MPI layer optimizations with CUDA IPC presented in Section III-C, we re-ran our training experiments and took scaling data up to 128 Lassen nodes (512 GPUs). The optimized results are depicted below in Figures 12 and 13. Furthermore, we applied hvprof to demonstrate the benefits of MPI-Opt by profiling 100 training steps under default MPI and MPI-Opt. The improvement for MPI_Allreduce is depicted in Figure 14 and Table I. **We demonstrate a 45.4% improvement in allreduce over the default**

The improvement for training throughput is depicted in Figure 12. **We demonstrate a 26% improvement in throughput over default MPI training.**

With MPI-Opt, we achieve vastly improved scaling up to 512 GPUs, above 70% scaling efficiency, and a 15.6% increase in scaling efficiency compared to default MPI. These results demonstrate the feasibility and benefits of our proposed approach.

Message Size (Bytes)	Time (ms)		Percentage Improvement
	Default	Optimized	
1-128 KB	392.0	391.2	≈ 0
128 KB - 16 MB	320.7	342.4	≈ 0
16 MB - 32 MB	1321.6	619.6	53.1
32 MB - 64 MB	5145.6	2587.151	49.7
Total Time	7179.9	3918.5	**45.4**

TABLE I: Allreduce time performance improvement

VIII. Key Insights

We present our key insights as follows:

- Horovod support for existing models is feasible, and an abundance of documentation makes this step approachable, even if the model is a novel DNN architecture (e.g. EDSR).
- While default Horovod support provides acceptable scaling efficiency for a small number of nodes, efficiency quickly drops off at larger scales. Default Horovod does not always perform optimally on novel DNN and system architectures.
- Profiling Horovod at the communication backend layer is vital to finding potential bottlenecks at larger scales.
- In order to achieve state-of-the-art performance at large scales, advanced CUDA-Aware MPI designs such as registration caching and CUDA IPC must be enabled.

We believe that this work provides general insights for other compute and communication-intensive DNNs that could benefit from distributed training on HPC systems. Figures 12 and 13 illustrate these insights for distributed training with

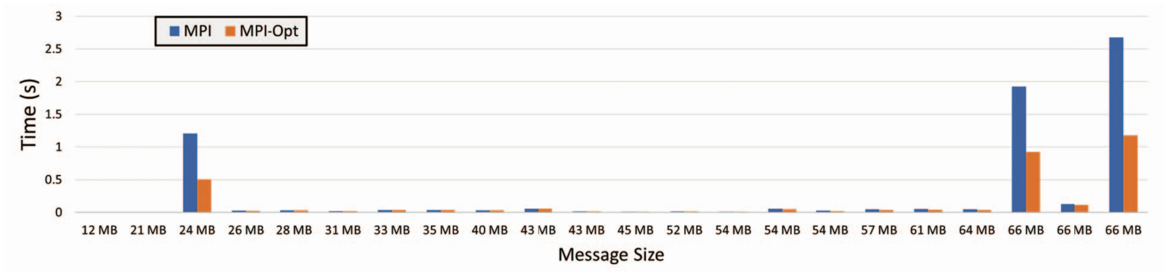


Fig. 14: Hvprof allreduce training profile for 100 training steps of EDSR on 4 GPUs

novel DNN architectures (in particular single-image super-resolution).

IX. Related Work

There exists several studies that involved distributed scalable DL training in the field of computer vision. While the optimization approach taken in [7] can be sufficient for DL frameworks that respect GPU mappings, the larger average message size for MPI_Allreduce required by EDSR are unable to be resolved with tuning at the Horovod layer alone. Zhang et al. [8] proposed a new distributed DL model based on self-attention mechanism and residual scaling. The novel model was then used to perform Remote Sensing (RS) image super-resolution tasks. In order to perform training on large volumes of Sentinel-2 data, they featured synchronized data parallelism and used Horovod to aggregate and average gradients over multiple workers. They also evaluated their model against DSen2 [25], another state-of-the-art CNN to perform upsampling on Sentinel-2 data. They were able to scale the training up to 16 GPUs while still achieving state-of-the-art performance. Kurth et al. [26] in their work that extended the DeepLabv3+ and Tiramisu segmentation models to serve as a direction towards climate analytics explored the possibilities of scaling DNN training across large number of GPUs (27,360). To achieve this, they coupled TensorFlow with Horovod support using Data Parallelism and implemented a hierarchical allreduce structure. LBANN is a DL framework that supports distributed training described by Jacobs et al [27], which managed to scale up Deep Generative models on scientific datasets. In their approach, a mix of data parallelism and model parallelism is adopted to scale DNN training up to 1,024 GPUs. A 109% parallel efficiency is achieved with 64 deployed trainers. According to CosmoFlow [28], it used TensorFlow and CPE ML plugin to provide support for DNN physical model training on 8,192 KNL nodes and reached 77% parallel efficiency.

X. Conclusion

The computational workloads for Deep Learning are rapidly increasing for emerging image processing applications such as autonomous driving, automatic medical image diagnosis [29], and climate analysis [26]. The extreme computation and communication requirements of these applications provide an excellent opportunity for distributed DNN training. We demonstrate that scaling deep learning super-resolution (DLSR) models to HPC systems is both feasible and approachable for existing single-node DNN implementations. Our optimization method achieves a percent improvement in scaling efficiency by 15.6%, which translates to a $1.26\times$ speedup in training performance. We believe that these results pave the way for efficiently training DLSR models and other novel DNNs that require long training times.

Acknowledgement

This research is supported in part by NSF grants 1818253, 1854828, 1931537, 2007991, 2018627, and XRAC grant NCR-130002.

References

[1] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, “S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters,” in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP ’17. ACM, 2017, pp. 193–205.

[2] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, “Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes,” *CoRR*, vol. abs/1807.11205, 2018. [Online]. Available: http://arxiv.org/abs/1807.11205

[3] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman, “Mesh-tensorflow: Deep learning for supercomputers,” *CoRR*, vol. abs/1811.02084, 2018. [Online]. Available: http://arxiv.org/abs/1811.02084

[4] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumoto, T. Tabaru, A. Ike, and K. Nakashima, “Yet another accelerated SGD: resnet-50 training on imagenet in 74.7 seconds,” *CoRR*, vol. abs/1903.12650, 2019. [Online]. Available: http://arxiv.org/abs/1903.12650

[5] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, “Enhanced deep residual networks for single image super-resolution,” 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[7] Q. Anthony, A. A. Awan, A. Jain, H. Subramoni, and D. K. D. Panda, “Efficient training of semantic image segmentation on summit using horovod and mvapich2-gdr,” in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 1015–1023.

[8] R. Zhang, G. Cavallaro, and J. Jitsev, “Super-Resolution of Large Volumes of Sentinel-2 Images with High Performance Distributed Deep Learning.” IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Online event (Online event), 26 Sep 2020 - 2 Oct 2020, Sep 2020. [Online]. Available: https://juser.fz-juelich.de/record/888525

[9] A. A. Awan, A. Jain, C.-H. Chu, H. Subramoni, and D. Panda, "Communication Profiling and Characterization of Deep Learning Workloads on Clusters with High-Performance Interconnects," in *Hot Interconnects 26 (HotI '19)*, August 2019.

[10] A. A. Awan, J. Bedorf, C.-H. Chu, H. Subramoni, and D. Panda, "Scalable Distributed DNN Training using TensorFlow and CUDA-Aware MPI: Characterization, Designs, and Performance Evaluation," in *The 19th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGRID 2019)*, May 2019.

[11] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. New York, NY, USA: ACM, 2017, pp. 193–205. [Online]. Available: http://doi.acm.org/10.1145/3018743.3018769

[12] A. Sergeev and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[13] M. Cho, U. Finkler, S. Kumar, D. S. Kung, V. Saxena, and D. Sreedhar, "Powerai DDL," *CoRR*, vol. abs/1708.02188, 2017. [Online]. Available: http://arxiv.org/abs/1708.02188

[14] NVIDIA, "NVIDIA Collective Communication Library (NCCL)," https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/index.html, 2016, Accessed: February 26, 2021.

[15] Z. Wang, J. Chen, and S. C. H. Hoi, "Deep learning for image super-resolution: A survey," 2020.

[16] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 1122–1131.

[17] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[18] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K. Lee, X. Wang, Y. Tian, K. Yu, Y. Zhang, S. Wu, C. Dong, L. Lin, Y. Qiao, C. C. Loy, and Q. Guo, "Ntire 2017 challenge on single image super-resolution: Methods and results," 07 2017, pp. 1110–1121.

[19] "NVIDIA GPUDirect RDMA," Accessed: February 26, 2021. [Online]. Available: http://docs.nvidia.com/cuda/gpudirect-rdma/

[20] S. Potluri, H. Wang, D. Bureddy, A. K. Singh, C. Rosales, and D. K. Panda, "Optimizing mpi communication on multi-gpu systems using cuda inter-process communication," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012, pp. 1848–1857.

[21] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. D. K. Panda, "Nv-group: Link-efficient reduction for distributed deep learning on modern dense gpu systems," in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi-org.proxy.lib.ohio-state.edu/10.1145/3392717.3392771

[22] J. Liu, J. Wu, and D. K. Panda, "High performance rdma-based mpi implementation over infiniband," *International Journal of Parallel Programming*, vol. 32, no. 3, pp. 167–198, 2004.

[23] H. Meur, E. Strohmaier, J. Dongarra, and H. Simon, "TOP 500 Supercomputer Sites," http://www.top500.org, 1993, [Online; accessed February 26, 2021].

[24] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, https://mvapich.cse.ohio-state.edu/, 2001, [Online; accessed February 26, 2021].

[25] C. Lanaras, J. M. Bioucas-Dias, S. Galliani, E. Baltsavias, and K. Schindler, "Super-resolution of sentinel-2 images: Learning a globally applicable deep neural network," *CoRR*, vol. abs/1803.04271, 2018. [Online]. Available: http://arxiv.org/abs/1803.04271

[26] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, Prabhat, and M. Houston, "Exascale deep learning for climate analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 51:1–51:12. [Online]. Available: https://doi.org/10.1109/SC.2018.00054

[27] S. A. Jacobs, B. V. Essen, D. Hysom, J.-S. Yeom, T. Moon, R. Anirudh, J. J. Thiagaranjan, S. Liu, P.-T. Bremer, J. Gaffney, T. Benson, P. Robinson, L. Peterson, and B. Spears, "Parallelizing training of deep generative models on massive scientific datasets," 2019.

[28] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook, K. Maschhoff, J. Sewall, N. Kumar, S. Ho, M. F. Ringenburg, P. Prabhat, and V. Lee, "Cosmoflow: Using deep learning to learn the universe at scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2018, pp. 819–829.

[29] A. Jain, A. A. Awan, A. M. Aljuhani, J. M. Hashmi, Q. G. Anthony, H. Subramoni, D. K. Panda, R. Machiraju, and A. Parwani, "Gems: Gpu-enabled memory-aware model-parallelism system for distributed dnn training," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.