2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 979-8-3503-0119-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/CCGRID57682.2023.00044

ScaMP: Scalable Meta-Parallelism for Deep Learning Search

Quentin Anthony, Lang Xu, Aamir Shafi, Hari Subramoni and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, *The Ohio State University*
{anthony.301, xu.3304, shafi.16, subramoni.1, panda.2}@osu.edu

***Abstract*—Deep Learning (DL) models are growing exponentially and require increasingly powerful High Performance Computing (HPC) systems to train them. Achieving state-of-the-art results requires carefully tuning the DL model architecture and training settings, which is a time-consuming process commonly relegated to distributed search frameworks and trial-and-error. However, search frameworks don't provide a flexible parallelism scheme within and among the chosen DL framework for modern out-of-core DL models.**

In this paper, we propose Scalable Meta-Parallelism for Deep Learning Search (ScaMP): a distributed Hyperparameter Optimization (HPO) and Neural Architecture Search (NAS) framework that supports out-of-core models with flexible parallelism schemes. SCaMP is integrated into the modern DL ecosystem, and enables both efficient parallel training of concurrent candidate architectures and aggregate device memory saturation via a powerful load balancing engine. SCaMP estimates the memory requirements of each candidate architecture and automatically applies the appropriate model-parallel degree and maximum batch size supported for the given candidate. Further, HPO and NAS with SCaMP are highly customizable via flexible configuration options.

We evaluate the benefits of our designs on synthetic training benchmarks and in training a state-of-the-art vision transformer model. We select transformers as a candidate DL model type and demonstrate a 29% improvement in end-to-end HPO time on 32 V100 GPUs on the Lassen and ThetaGPU HPC systems. Further, we demonstrate a reduction in the proportion of NAS time spent in communication from 28% to 15%. Finally, we thoroughly verify the correctness of SCaMP by training a state-of-the-art SwinIR model.

***Index Terms*—Neural Networks, DNN, MPI, GPU**

I. Introduction

Deep Learning (DL) methods continue to provide state-of-the-art results across a variety of data formats with a recent focus on text processing [1], [2], [3] and image processing [4], [5], [6], [7] applications. In a nutshell, DL is a subset of Machine Learning (ML) that uses Deep Neural Networks (DNNs) to learn implicit relationships between input and output data. DNNs consist of a matrix of weights that are first updated in order to minimize prediction loss before applying the final trained model to a dataset. In some cases such as deep learning recommendation models [8], DL models can be composed of multiple DNNs combined with other data processing schemes (such as embedding tables). DL training is highly compute-intensive, and requires massive amounts of data. Most application improvements are achieved by improving the dataset [9], [10], [11], [12], model architecture [4], [5], [13], and training environment [14], [15], [16], [17] (e.g. optimizers, parallelism schemes, etc). In recent years, the scale of DL models and datasets have exploded to hundreds of billions of parameters [2] and terabytes of data [12].

As a result, distributed DL has become the standard training method for new DL models. As the largest models grow from hundreds of millions [4] to hundreds of billions of parameters [2], new parallelization schemes have arisen to efficiently train DL models across thousands of processors [18], [19], [20], [8], [21]. We denote models that fit within device memory as *in-core* models, and models that require more memory than a single device as *out-of-core* models. **While many prior works have explored parallelism strategies within a single DL architecture, few have proposed schemes that efficiently parallelize the search process over multiple possible DL architectures or training settings.** Significant work has been performed towards reducing the communication overheads of each scheme, yet the proportion of training time spent in communication remains high. Further, such schemes are only useful for training a single DL model with a single set of training parameters at a time. We propose shifting the HPO/NAS parallelism focus from optimizing data parallelism to optimizing subnet[1] placement such that communication is minimized.

Modern large DL models are composed of many possible settings for training settings (e.g. optimizer settings such as learning rate, weight initialization, severity of weight and learning rate decay, number of warmup iterations, etc). Therefore, a large number of candidate models must be partially trained before the final model can be trained start-to-finish. This process is known as **hyperparameter optimization**. As models require ever-increasing computational resources to train, evaluating candidate models has also become computationally expensive. Another common DL training workflow is **neural architecture search**, where details on the underlying DL architecture itself (e.g. the width and depth of feed-forward layers, number of attention heads in transformer networks, etc) is left to an automated optimization process by searching through candidate models and evolving them to a model with better performance. Further, candidate architectures commonly

This research is supported in part by NSF grants 1818253, 1854828, 1931537, 2007991, 2018627, 2112606, and XRAC grant NCR-130002.

[1]use the term *subnet* to denote a single architecture and training setting combination

979-8-3503-0119-9/23/$31.00 ©2023 IEEE
DOI 10.1109/CCGrid57682.2023.00044

require different amounts of device memory (e.g. changing the hidden size or number of attention heads in a transformer model), and it's common to have a set of subnets with multiple instances of crossing the device memory boundary. Currently, the user must empirically determine which subnets lead to out of memory errors and manually set their model parallel settings before the full sweep can be run to completion. This is extremely tedious and error-prone. **We wish to estimate the amount of memory required to instantiate a given subnet, and automatically choose both the model-parallel degree required to fit within memory, and the maximum batch size that fits within memory.**

A. Motivation

The key motivator for this work is in reducing unnecessary data-parallel communication for any DL training scheme that requires the training of multiple subnets (e.g. HPO and NAS). Most existing meta-parallel frameworks and techniques rely upon training one or very few subnets at once, and scale to many processors with simple data parallelism. While the use of data parallelism reduces the training time significantly, ideally by a factor of the number of devices d, it incurs a non-trivial communication overhead. We believe that thoughtfully placing concurrent subnets across parallel hardware will nearly eliminate the communication overhead of data-parallelism while still reducing the overall subnet training time by a factor of d.

As a secondary motivator, many existing subnet training frameworks do not consider the case of crossing parallelism boundaries within a subnet sweep (i.e. for what subnet architectures within the sweep does the model require model- or hybrid-parallelism?). By estimating the memory requirements of each subnet before training, we wish to choose the degree of parallelism before starting the training sweep. This functionality will avoid the significant programmer effort required by empirically determining which subnets require model- or hybrid-parallelism, then statically choosing the parallelism settings for those subnets.

B. Problem Statement

Modern DL training pipelines commonly involve one of two subnet training schemes: Hyperparameter Optimization (HPO) or Neural Architecture Search (NAS). HPO involves finding an optimal set of hyperparameters (e.g. optimizer settings, batch size, number of epochs, etc). While a number of distributed hyperparameter search libraries exist, there are two shortcomings: models are typically trained inefficiently in sequence with data parallelism, and HPO frameworks rarely support out-of-core models that require advanced model-parallel techniques. NAS is the problem of finding the highest-performing DNN architecture on a given training scheme (i.e. the dataset and hyperparameters are static while the architecture is modified). Similar to HPO, NAS frameworks exist, but they share the same shortcomings as HPO. **We propose shifting the HPO/NAS parallelism focus from optimizing data parallelism to optimizing subnet placement.**

C. Challenges

There are two broad challenges addressed in this paper. The first broad challenge is: ***Can we reduce the time spent in data-parallel communication by thoughtfully placing concurrent subnets?***. We seek to create a flexible yet efficient framework that extends to many subnet architectures, sizes, and system scales. To answer this broad question, we solve the following concrete challenges:

- How should a framework load-balance a set of subnet architectures such that the time spent in communication is minimized?
- How do we expect these meta-parallelism strategies to behave analytically?
- What benefits can efficient meta-parallelism strategies provide to reduce overall HPO/NAS training time?

The second broad challenge we resolve is: ***Can we avoid empirically determining which subnets don't fit within device memory by estimating the memory requirements of each subnet, automatically choosing the required degree of model-parallelism and maximum batch size?***. We seek to incorporate a robust memory estimation engine to enable SCaMP to automatically handle subnets of varying memory overheads.

- Can the memory requirements of a given subnet be determined before training begins? Can this information be used to automatically determine the degree of model-parallelism required to fit within device memory, as well as the maximum batch size?
- Given that subnets often have differing memory requirements that may require varying degrees of model parallelism, how should these subnets be efficiently load-balanced and managed by a unified interface?
- Can subnets of varying model-parallelism settings still be load-balanced efficiently at scale?

D. Proposed Solution

We believe that a flexible parallelism suite for training multiple distributed subnets in parallel will alleviate these shortcomings. We define parallelism schemes of distributed subnets as *meta-parallelism*, and explore its properties. Specifically, we introduce and evaluate **Scalable Meta-Parallelism for Deep Learning Search (SCaMP)**. SCaMP is a flexible and efficient meta-parallel framework for any DL problem involving subnets (e.g. HPO, NAS, parallel inference). SCaMP monitors and manages subnets in real-time for the end-user, and is incorporated with modern DL software such as DeepSpeed [20].

E. Contributions

Our contributions are as follows:

C1) We proposed, designed, and evaluated SCaMP: a scalable framework for parallel HPO and NAS specifically targeted for transformer models and HPC systems (Section III)

C2) We modeled and explored the tradeoffs of a variety of meta-parallelism strategies, and demonstrate why tuning

each parallelism method leads to significant end-to-end hybrid parallel subnet training time reductions. (Section III)

C3) Vastly improve user productivity for HPO and NAS workloads by automatically estimating the device memory requirements of each subnet setting, therefore allowing the model-parallel degree and maximum batch size to be set automatically for a given subnet (Section III-6)

C4) Thoroughly verify the correctness of the SCaMP framework by training a model to state-of-the-art accuracy. (Table II)

C5) Create an analytical model for meta-parallelism strategies (Section IV) and compare against experimental data (V)

C6) Evaluate SCaMP's performance on large-scale HPC systems. Report up to 29% reduction in overall training time over purely data-parallel training for SwinIR and a reduction in the proportion of time spent in communication from 28% to 15% for Megatron. (Section V)

II. Background

A. Deep Learning

It is well-known that because of the huge amount of iterations included, training DNN models is highly demanding in computation and communication resources [22], [23], [24], [25]. As larger and deeper DNNs appear, there has been a growing need for distributed training support in order to scale out model training to more hardware. Traditional deep learning frameworks such as PyTorch and TensorFlow contain distributed training support. Recently, frameworks specifically targeting distributed training such as Horovod by Uber [26] and DeepSpeed by Microsoft [20] have appeared. A popular parallelism strategy to distribute DNN training is data parallelism. This scheme first copies an instance of the DNN model to each CPU/GPU. Then, the training data will be partitioned arcoss all CPUs/GPUs. The size of a batch of data that is sent to each CPU/GPU at each global training step is called the batch size. Data parallelism typically requires a synchronization at the end of each training step to average the gradients from all processes. This is always achieved by collective operations such as MPI_allreduce or NCCL. The averaged gradients will be sent back to each process and the DNN copies will proceed to the next global training step with the updated gradients. Model-parallelism is a broad term referring to any strategy that partitions the model itself across processors. We denote models that fit within device memory as *in-core* models, and models that require more memory than a single device as *out-of-core* models. Broadly, in-core models can be trained with data-parallelism and out-of-core models require model-parallelism. Note that multiple data-parallel instances of model-parallel models is the most common method of training out-of-core models.

B. Deep Learning Search

As modern DL models become more complex, efficient methods for searching over possible neural architectures have become enormously important. Manual designs of neural architectures are more time-consuming and often subject to errors. Neural Architecture Search (NAS) is an efficient method of automating neural architecture engineering and a subfield of AutoML [27]. Several DNNs built with NAS have outperformed traditional manually crafted neural architectures in fields such as image classification [28], [29] and object detection [28]. NAS is often characterized by three dimensions: **Search Space**, **Search Strategy**, and **Performance Estimation Strategy**. NAS typically features a process that aims to discover and evaluate candidate networks in a well-defined search space through an efficient search strategy. Then the candidate will be evaluated using performance estimation strategy and compared to previous candidates for final selection. A NAS search space defines a space where all eligible candidate networks reside. The search strategy draws the path through which we discover candidate networks in a search space. The performance estimation strategy is the method of evaluating a candidate's performance on unknown data. NAS methods are generally requiring much computation resources since within a large search space, there are always a large number of candidates to be searched, trained and evaluated. There have been much efforts in reducing search space complexity [27].

Apart from NAS, Hyper-parameter optimization (HPO) is also an important compontnet of AutoML in finding optimal hyper-parameter combinations for neural network architectures and model training process. Hyper-parameters are parameters that are cannot be updated during the training of a DNN model. Typical hyper-parameters include the number of hidden layers, activation function, learning rate, batch size, optimizer and so on. HPO is often viewed as the final step of model design and the first step of actually training a network. In the past, tuning hyper-parameters are heavily dependent on human expertise and are set manually before the training process begins. The process of HPO aims at removing preset humen experience from a deep learning cycle. However, the automation is at a cost of extra computational resources. Currently, HPO faces mainly three challenges [30]: To reduce the work load of human expertise in tuning networks and smooth the research and development process; To enhance model quality through efficient training strategies [31]; and to output optimal hyper-parameter sets that are reproducible and rational [32].

C. Image Super-Resolution

Image super-resolution has been a long-lasting research topic in computer vision. It aims at generating one or more high-resolution image from its low-resolution counterparts. It welcomes various applications including ultrasound imaging [33], security [34], [35] and media content enhancement etc. However, image SR has been challenging since a single low-resolution image can be mapped towards several possible high-resolution candidates based on different approaches. Traditional approaches include prediction-based methods [36], [37] and statistical methods [38], [39]. In recent years, we have witnessed the rising of deep learning methods that create possibilities for various learning-based methods. Starting with several inspiring precursors [40], [41], [42], [43], convolu-

tional neural networks (CNN) has been a popular backbone for image super-resolution with most adoptions focusing on architecture designs such as residual learning [44] and dense network connections [42]. Recently, Generative Adversarial Nets (GAN) have also staged promising results [45].

III. Design

We designed and implemented SCaMP: a scalable distributed meta-learning framework powered by DeepSpeed, and applicable to transformer architectures including SwinIR. There are several key features of SCaMP depicted in Figure 1(b), which are:

- User Configurations: These configurations contain information on the NAS settings (block architecture, #subnets, #subnet architectures, etc), model training settings (#epochs, batch size, learning rate, etc), and distributed training settings (i.e. parallelism and sharding settings for DeepSpeed).
- Controller: The controller manages training by launching and monitoring subnet training jobs within training nodes.
- Load Balancer: A powerful estimation engine that ensures aggregate device memory is saturated while necessary communication is minimized.
- Trainer: A lightweight interface built atop DeepSpeed to help the Controller launch and manage training jobs.

At a high level, a typical SCaMP workflow follows these steps (see Figure 1(b)):

1) The user writes a configuration and passes it to SCaMP, which launches a controller.
2) The controller instantiates the model and feeds the user configuration settings into the load balancer
3) The load balancer estimates the model's memory, chooses the appropriate subnet placement and batch sizes, and generates a set of parallel experiments.
4) The controller then takes this experiment set and launches/monitors them on parallel hardware via the trainer interface.

The controller keeps detailed logs on each subnet run, and prepares a final summarization on the throughput and loss curves for each subnet. Once the subnets have completed training on however many iterations the user configuration dictates, the user reads through the summarization and associated logs and chooses the best candidate subnet and hyperparamter combination to fully train with a framework that is external to SCaMP. Each component is described in detail in the following sections:

1) User Configurations

We seek to maximize user control over the NAS/HPO process while minimizing their required knowledge of the underlying framework. Therefore, we adopt the configuration approach and allow users to tune as many clearly-documented knobs as possible without modifying several complex training files to meet their specific NAS/HPO training job.

A given user configuration is split into a few key parts:

- **Settings for the overall meta-learning job.** These include settings such as the number of subnets, what range of hyperparameters they're trained over, what search space to use, etc.
- **Model training settings.** These include how many training steps each subnet is trained for, what values to use for static hyperparameters, training data and dataloader options, etc.
- **Distributed settings.** These optional settings can determine the constraints for SCaMP's load balancer, including the number of nodes/GPUs, how many concurrent subnets to train at once, etc. If the user wishes to override the load balancer and instead decide how to map subnets, they may pass custom mappings of subnets to GPUs.

Listing 1 contains a snippet of an example configuration file.

```
# SCaMP Sample User Config

{
   # Distributed Training Settings
   # Choose the number of subnets to parallelize
   "num-parallel-subnets": 4
   # Optionally map each subnet to a pair of "node
    :[list of GPU IDs]
   "subnet-mappings": {0:[0,1], 0:[2,3], 1:[0,1],
    1:[2,3]}
   ...
   # NAS or HPO Settings
   "num-subnets": 1000,
   "strategy": "evolutionary",
   "hidden_size": [1028, 2048, ...],
   ...
   # Model Settings and static hyperparameters
   "scale": 2,
   "n_channels": 3,
   ...
   # Training Settings and static hyperparameters
   "train_micro_batch_size_per_gpu": 4,
   "num_epochs": 10,
   "log-interval": 10,
   "checkpointing": True
      "optimizer": {
     "type": "Adam",
     "params": {
       "lr": 0.0002,
     ...
     }
   },
   "fp16": true,
   "zero_optimization": {
    "stage": 1,
  ...
  },
   ...
}
```

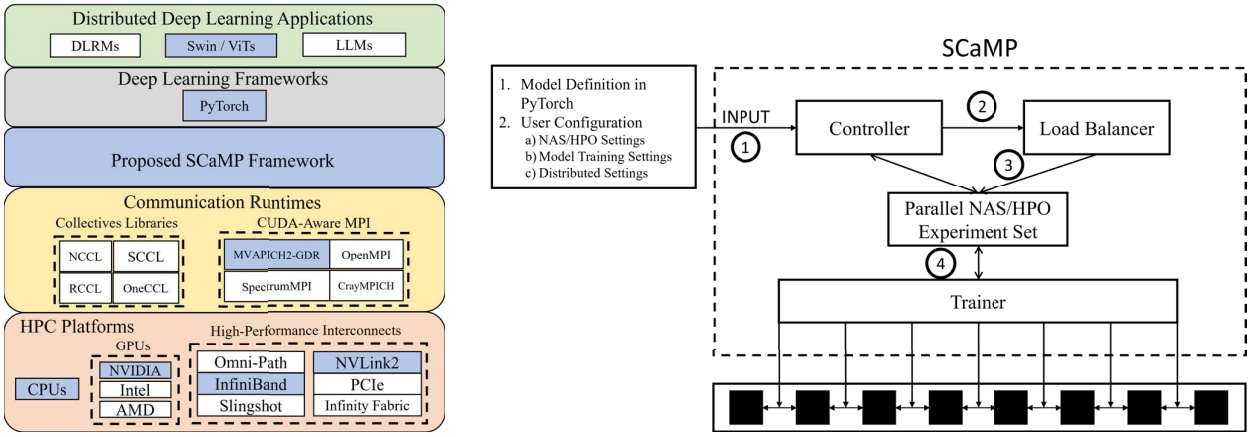
Listing 1. Example of user configuration in SCaMP

2) Memory Estimation

Estimating the memory of a given subnet configuration. Specifically, we seek to estimate the GPU memory M required to fit a model during training.

$$M_{Tot} = M_{\mathrm{m}} + M_{\mathrm{o}} + M_{\mathrm{g}} + M_{\mathrm{a}} \tag{1}$$

Where M_{m} is the memory necessary to fit the model, M_{o} for the optimizer (we assume mixed-precision Adam in this work), M_{g} for the gradients, and M_{a} for the activations. Given the number of model parameters p, DeepSpeed ZeRO stage z (ranges from $0-3$[18]), number of bytes per parameter k



(a) SCaMP Software Stack

(b) SCaMP Architecture

Fig. 1. SCaMP is a scalable and flexible framework for meta-parallelism composed of a Controller (Section III-3), a load balancer (Section III-4), and a trainer (Section III-5) that directly interfaces with the parallel hardware.

($k = 2$ for fp16, $k = 4$ for fp32), devices d, tensor parallel degree t (ranges from $1-d$, and $t = 1$ if no tensor parallelism is used), sequence length s, batch size b, hidden size h, number of transformer layers L, the approximate memory overhead in bytes required by a given model configuration is given by Equations 2, 3, 4, and 5 below.

$$M_{\mathrm{m}} \approx \begin{cases} \frac{k \cdot p}{d \cdot t}, & z = 3 \\ \frac{k \cdot p}{t}, & \text{else} \end{cases} \quad (2) \qquad M_{\mathrm{o}} \approx \begin{cases} \frac{6k \cdot p}{d}, & z \geq 1 \\ 6k \cdot p, & \text{else} \end{cases} \quad (3)$$

$$M_{\mathrm{g}} \approx \begin{cases} \frac{k \cdot p}{d}, & z \geq 2 \\ k \cdot p, & \text{else} \end{cases} \quad (4) \qquad M_{\mathrm{a}} \approx sbhL(10 + \frac{24}{t}) \quad (5)$$

Note that: (1) while the equations for M_{m}, M_{o}, and M_{g} are specific to DeepSpeed, mixed-precision Adam and transformer architectures, they can easily be extended to other architectures and optimizers within the SCaMP framework. (2) The activation memory equation M_{a} is specific to a transformer using a selective activation checkpointing scheme like that used in Megatron[46]. For future work, we intend to extend these equations to support other optimizers, activation checkpointing schemes, and DNN architectures common in distributed DL.

3) Controller

Standard NAS implementations train each subnet architecture via data-parallelism (e.g. Figure 2(d)), and therefore inherit the communication overhead from data parallelism [47]. Instead, SCaMP supports training multiple subnet architectures at once by specifying the desired mapping within the user configuration. Consider the difference between Figures 2(a) and 2(d). Training a single subnet at a time via data-parallelism (i.e. Figure 2(d)) will suffer from communication overhead, but if the model fits in memory we can assign a single subnet to each GPU **and completely remove the need for communication operations among GPUs.** The controller launches a distinct training job on each GPU, and monitors their completion. Note that distributed overhead is small but not zero because each NAS training iteration is synchronous (i.e. all subnets must complete training and evaluating this iteration's subnets before proceeding). Once all subnets within a NAS iteration have been trained and evaluated, the new subnet training jobs are assigned across GPUs. Note that this scheme requires the number of subnets to equal the number of total GPUs. In order to support any number of subnets, any intermediate parallelism scheme (e.g. data-parallelism of a single subnet within each node as in Figure 2(c)) is available by specifying the desired mapping in a user configuration file. Finally, SCaMP inherits DeepSpeed's ability to split large models that cannot fit inside a single GPU's memory via three-dimensional parallelism (e.g. Figure 1(b)).

TABLE I
NUMBER OF SUBNETS TRAINED CONCURRENTLY FOR EACH EXAMPLE PARALLELISM SCHEME IN SCAMP

Parallelism Scheme	Number of Concurrent Subnets
Full DP	1
Node DP	$\frac{\text{Number of GPUs}}{(\text{GPUs/Node})}$
Half-node DP	$2 \times \frac{\text{Number of GPUs}}{(\text{GPUs/Node})}$
NP	Number of GPUs

4) Load Balancer

The load balancer determines how to place subnets so that resources are maximally used while minimizing data parallelism within subnets. Depending on the number of subnets, the number of nodes available and the number of GPUs available, the load balancer seeks to decide a subnet parallelism strategy such as those in Figure 2 to minimize data-parallel communication. A key feature within the load balancer is the ability to estimate the memory overhead of each subnet. Specifically, it generates an array where each element contains the memory overhead necessary to fit a single replica of that subnet $M_i(model\_def, datatype)$. Then the load balancer iterates over this array, and determines the minimum number of devices required to train each subnet

without data parallelism with the maximum batch size. Given that total available GPU memory is given by G_M, the memory required by subnet i is M_i, batch size is given by B, and the memory per batch is given by M_B, we seek to find the minimum number of devices d_i required to train a given subnet while maximizing GPU memory utilization via the batch size.

$$d_i(G_M, M_i) = \lceil \frac{M_i}{G_M} \rceil + \max_B \{B * M_B | B * M_B < G_M\} \quad (6)$$

These memory estimation equations are combined with the load balancer (discussed in Section III-4) to automatically fit the maximum batch size for a given user configuration. Note that when $d_i = 1$, the load balancer will seek to evenly distribute resources to each subnet for data parallelism, maximize the batch size for each data parallel instance, and as nearly approximate the *No Parallelism* meta-parallelism scheme in Figure 2 and Table I. When $d_i > 1$, however, the load balancer must apply model parallelism to the model, maximize the batch size, and evenly distribute resources to each subnet for data parallelism. Ideally, the resulting meta-parallel strategy will still approximate the *no-parallelism*. An example of a load-balanced workload for 4 subnets, each roughly doubling in size, is depicted in Figure 3. If the load balancer determines that the number of subnets requires more GPUs than are available, which is often the case, then training is split into a number of steps, each step containing a load-balanced set of concurrent subnets.

5) Trainer

The trainer is provided two inputs:

- The controller propagates the user's configuration file and information on the state of the system (e.g. number of nodes, GPUs per node, etc).
- The load balancer provides a mapping of each subnet to the number of GPUs necessary and the best batch size to use.

The controller launches training jobs on target GPUs via the Trainer. In order for the controller to log and synchronize training jobs, a lightweight API between the controller and underlying distributed DL training framework (DeepSpeed) was developed. The full API is depicted in Listing 2. If any subnet training jobs fail, these failures are summarized and reported to the user independently of the overall HPO/NAS workload to avoid unnecessary slowdowns.

```
# Maintains overall training information such as the best subnets, iteration number, etc
class SCaMP_Status

# Stores the parsed configuration file
class SCaMP_Args config

# Returns subnet training status (e.g. "training", "complete", "failed")
status(int gpu_id)

# Launches a DeepSpeed training job
launch(List<int> gpu_ids, SCaMP_Args config)

# Evaluates trained subnets and updates status
evaluate(List<int> gpu_ids, SCaMP_Status status)

# Updates subnet architectures and updates status
update_parameters(SCaMP_Args config, SCaMP_Status status)

# Synchronizes each subnet step
synchronize(List<int> gpu_ids)

# Displays status to user
log(List<int> gpu_ids, SCaMP_Status status)
```

Listing 2. SCaMP API

6) Meta-Parallelism Strategies Crossing Parallelism Boundaries

It's often the case that the model size itself is modified (either directly or indirectly) by hyperparameter settings. We wish to account for the case that the model becomes large or small enough to cross a model-parallelism boundary (e.g. when increasing a hidden dimension hyperparameter, increase the model-parallel degree m if the model is estimated to run out of memory).

Without meta-parallel techniques, the user would need to 1) empirically determine which hyperparameter settings correspond to a given model-parallel degree, 2) split the overall hyperparameter search into distinct sets, one for each model-parallel degree. We have added the flexibility within SCaMP to estimate the memory requirements for a given hyperparameter set and maximally use available resources on-the-fly.

By estimating the memory requirements of model-parallel subnets on-the-fly and placing them on a the minimal number of GPUs possible, SCaMP is able to vastly decrease the time spent in data parallel communication. Weakening the reliance on data parallel communication greatly improves the scalability of meta-parallel schemes such as hyperparameter optimization and neural architecture search. By carefully placing model-parallel subnets on parallel hardware, SCaMP is again able to vastly reduce data parallel communication overhead and improve scalability

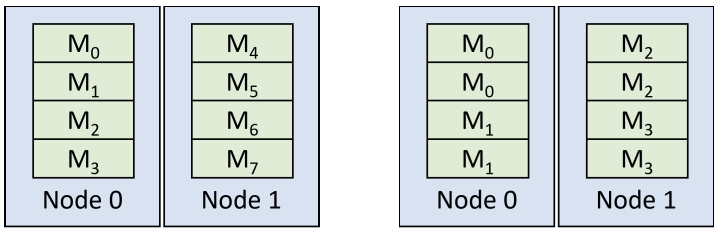
IV. Analytical Model for Meta-Parallelism

1) Basic Parallel Training

We present an anlytical model to estimate the potential speedup of a given meta-parallelism strategy. The number of processors used in a given training run are denoted by d. The computation and communication time are denoted by T_{comm} and T_{comp}. T_{comm} is itself split into model-parallel (T_{MP_comm}) and data-parallel (T_{DP_comm}) communication. We model T_{MP_comm} as independent of the total number of processes[2]. The time spent in data-parallelism could relate to the total number of devices d in a myriad of ways depending on its implementation, but will follow Eq 7 in the vast majority of cases, and often scales like $O(d)$ or $O(log(d))$ [48].

$$\begin{aligned} T_{DP\_comm}(d) &\propto d \\ T_{DP\_comm}(1) &= 0 \end{aligned} \quad (7)$$

[2]While the model-parallel communication cost does depend on the number of devices (e.g. the pipeline bubble decreases in size as the data-parallel degree increases), we will assume that any model-parallel trials are partitioned across the minimal number of devices to run



(a) No Parallelism (NP)

(b) Partially Data Parallel (Half-node DP)

(c) Node-boundary Data Parallelism (Node DP)

(d) Fully Data Parallel (Fully DP)

Fig. 2. Comparison of parallelization strategies in SCaMP. Each box within a node is a GPU, and M_n refers to training model subnet n on a given GPU. All subnets fit within a GPU and are approximately the same size.

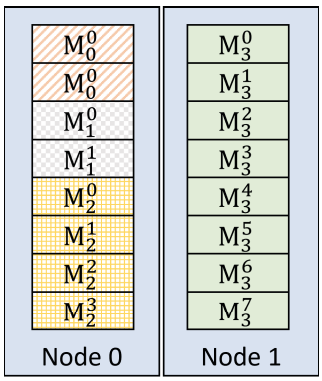


Fig. 3. Training across model parallelism boundaries. Each box within a node is a GPU, and M_n^k refers to training model-parallel partition k of model subnet n on a given GPU. The model size of each subsequent subnet is roughly doubled (e.g. subnet 0 fits on a single GPU, subnet 1 requires two GPUs, etc).

We can model the overall time spent in communication as

$$T_{comm}(d) = T_{MP\_comm} + T_{DP\_comm}(d) \tag{8}$$

We will operate under the basic assumption that the time spent in computation is inversely proportional to the data-parallel degree. Therefore, the baseline time to train a single subnet is given by Eq 9

$$T_{total}^{single}(d) = \frac{1}{d} * T_{comp} + T_{comm}(d) \tag{9}$$

2) Meta-Parallelism Strategies

In order to perform hyperparameter optimization or neural architecture search, a series of n subnets are first defined. Each subnet is trained up to a given number of steps, and the resulting model accuracy is recorded. The best subnet is then chosen for full training. Given that we have n subnets to evaluate, and that we wish to naively train them sequentially, the time it takes to train all subnets is given in Eq 10.

$$T_{total}^{Full-DP}(d) = \sum_{i=1}^{n}(\frac{1}{d} * T_{comp}^{i} + T_{comm}^{i}(d)) \tag{10}$$

This method of training wastes significant time in communication if the model fits within less memory than the aggregate device memory. Therefore, we wish to reduce the time spent in communication and increase the time spent in computation. To do this, we train c concurrent subnets which we call *Partial-node DP*. There are many possible subnet placement strategies available, some of which are depicted in Figure 2. The time to train n subnets is now given by Eq 11.

$$T_{total}^{Partial-DP}(d) = \sum_{i=1}^{n/c} \max_{j}\{(\frac{c}{d} * T_{comp}^{j} + T_{comm}^{j}(\frac{d}{c})) \; | (i-1) * c < j \leq i * c\} \tag{11}$$

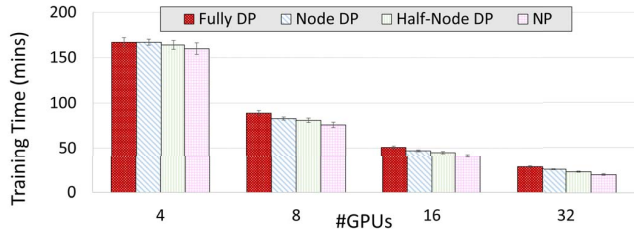
Where $0 \equiv c \mod d$. *No Parallelism (NP)* is given by Eq 11 with $c = d$, and *Full DP* is given by Eq 11 with $c = 1$. In all of these meta-parallel strategies, the time spent in computation is increased proportional to c while the time spent in communication is reduced proportional to c.

V. Performance Characterization

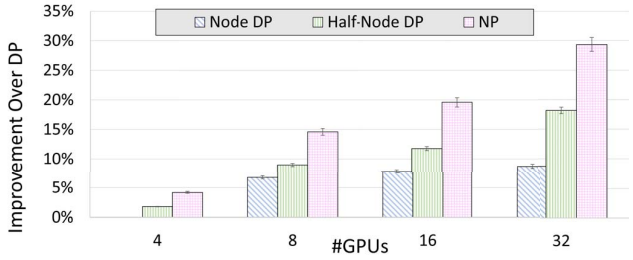
1) Node Architecture

All experimental evaluations[3] were conducted on the Lassen cluster at Lawrence Livermore National Laboratory and on the ThetaGPU cluster at Argonne Leadership Computing Facility [49]. ThetaGPU is comprised of 24 NVIDIA DGX A100 nodes. Each node is equipped with 2 AMD Rome CPUs, 1TB

[3]The choice of cluster for a given application was purely made out of external factors such as available compute and ease of software compatibility



(a) Time to train subnets in SCaMP using a variety of parallelism strategies and system scales



(b) Percent improvement in SCaMP parallelism strategies over pure data parallelism

Fig. 4. Performance of various parallelism strategies in SCaMP. Most previous HPO/NAS frameworks rely on *Fully DP* or *Node DP*, which are subject to significant data-parallel communication overheads. SCaMP is able to reduce the data-parallel communication overhead required to train a set of subnets by using meta-parallelism strategies such as those in Figure 2.

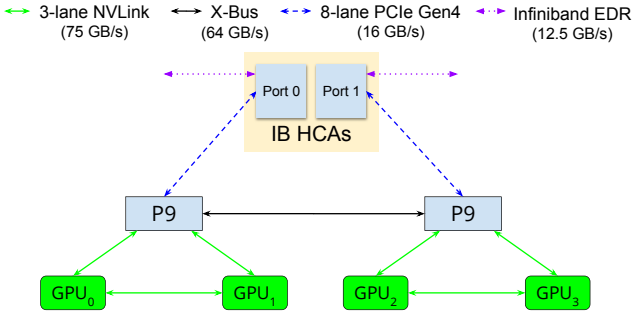


Fig. 5. Node topology of the Lassen supercomputer at LLNL

DDR4 memory, and 8 NVIDIA A100 Tensor Core GPUs. The NVIDIA DGX A100 GPU has 40GB HBM2, and is connected with the second generation NVIDIA NVSwitch. Each node is connected with Mellanox ConnectX-6 VPI HDR InfiniBand/Ethernet network adapters, and the overall cluster includes 20 Mellanox QM9700 HDR200 switches wired in a fat-tree topology. Lassen is the #34-ranked machine in the TOP500[50] as of December 2022 and consists of 792 GPU nodes each with four 16 GB memory NVIDIA Volta V100 GPUs. Lassen nodes contain two 44-core IBM Power 9 architecture CPUs, and are connected via Mellanox Infiniband EDR in a fat-tree topology

2) Software Libraries

We used MVAPICH2-GDR 2.3.7 [51] for all DL experiments. All backends and frameworks were built with CUDA 11.4.152 on ThetaGPU and CUDA 11.4.100 on Lassen. All micro-benchmark evaluations were carried out with OSU Micro-Benchmarks (OMB) 6.1. For our DL evaluations, we used source-built PyTorch v1.12.1 and DeepSpeed v0.7.4.

3) DL Training Settings

For vision models, we used a mixture of SwinIR[52] and vision transformers (ViT)[53]. We fully trained a NAS/HPO model based on SwinIR. Details and results are in Section V-A and Table II, respectively.

For text-based models, we used Megatron-DeepSpeed[54]. These models were used to evaluate model-parallel subnets. We modified only the tensor parallelism degree in SCaMP, keeping pipeline parallelism constant (we intend to extend the load balancer to exploit multiple degrees of parallelism in future work. See Section VII). All Megatron-DeepSpeed models were GPT-based and dense transformers.

4) Application Results

We first evaluated our distributed meta-learning framework (SCaMP) on a variety of system scales within the Lassen supercomputer (Node architecture in Figure 5). The results of training each of the parallelism strategies using ViT-based Megatron-DeepSpeed subnets are in Figure 2 with SCaMP on 4-32 V100 GPUs on Lassen are depicted in Figures 4(a) and 4(b). Note that No Parallelism (**NP**) refers to the parallelism strategy in Figure 2(a), Half-Node Data-Parallel (**half-node DP**) refers to Figure 2(b), Node Data-Parallel (**Node DP**) refers to Figure 2(c), and Fully Data-Parallel (**Fully DP**) refers to Figure 2(d).

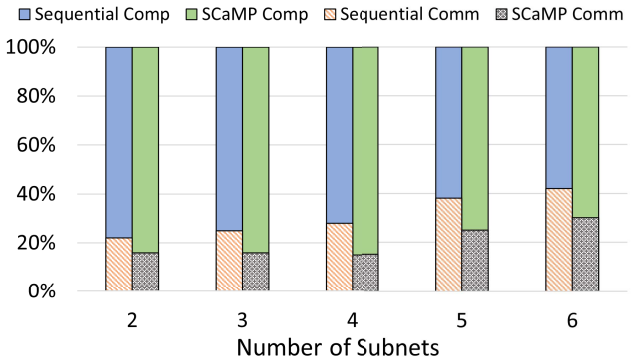


Fig. 6. Proportion of communication to computation for various meta-parallelism strategies. Each subsequent subnet is roughly double the size of the previous subnet (see Figure 3 for an example of 4 subnets). We denote the training of a single subnet at a time without load balancing as *sequential*.

The key takeaway of Figures 4(a) and 4(b) is that each parallelism strategy has a progressively less demanding data-parallel communication overhead. Consider the case at 8 GPUs. **Node DP** removes the inter-node communication present in **Fully DP**, **half-node DP** removes the X-Bus

communication present in **Node DP**[4], and **NP** removes the data-parallel communication overhead entirely, culminating in a 29% reduction in overall training time. Note that the meta-parallelism strategy is dependent on the number of concurrent subnets being trained at a given time. The exact number of concurrent subnets that can be trained for each meta-parallelism scheme is depicted in Table I.

Calling back to the motivation stated in Section I-A, SCaMP seeks to reduce the data-parallel communication overhead, and therefore shift the overall application time from communication to computation. Therefore, we profiled the time spent in each communication call and used this to calculate the overall proportion of time spent in computation and communication on the ThetaGPU system (8 A100 GPUs per node). We have evaluated the case where each subsequent subnet requires roughly double the memory to fit within memory (i.e. by doubling the number of layers, so that $2^{\text{num\_subnets}}$ GPUs is needed to fit all of the subnets in aggregate memory). An example of this scheme with 4 subnets is depicted in Figure 3. The resulting communication vs computation data both with SCaMP's load balancing engine versus training a single subnet at a time without load balancing using GPT-based Megatron-DeepSpeed subnets is depicted in Figure 6. The main takeaway from this result is that SCaMP's load balancing strategies greatly reduce the time spent in data-parallel communication, even if model-parallel communication is still required by larger subnet architectures. Another insight is that both SCaMP and *Sequential* require significantly more communication once the largest model-parallel subnet is split across nodes.

Finally, we wish to compare experimental results against the analytical model defined on IV. We first measure the resulting experimental error on Lassen with a static number of devices but across meta-parallelism strategies. These results are depicted in Figure 7. To ensure that the analytical model also holds at multiple system scales, we also measure the experimental error across a range of GPU scales. This result is depicted in Figure 8.

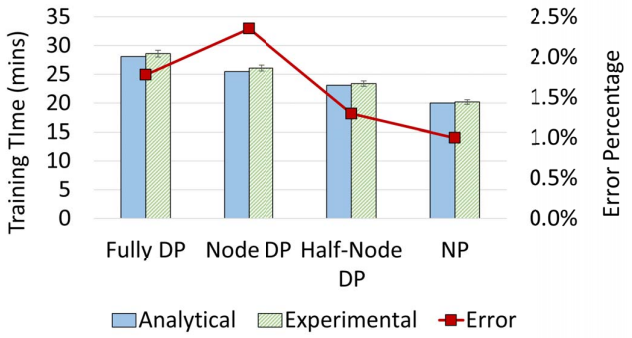


Fig. 7. Comparison of analytical model and experimental results for various meta-parallelism strategies at 32 V100 GPUs on Lassen

[4]Recall that pairs of GPUs are connected via NVLINK on Lassen. See Figure 5

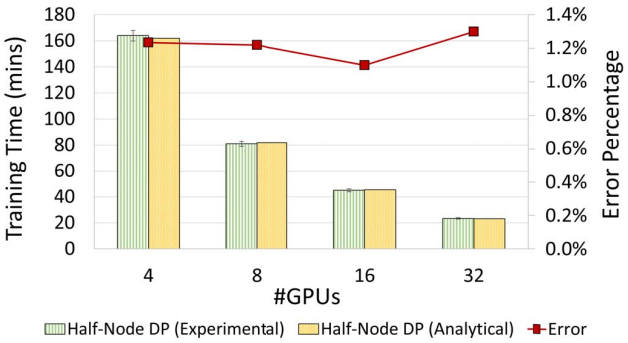


Fig. 8. Comparison against analytical model across GPU scales on Lassen for *half-node* (Figure 2) parallelism strategy

A. SwinIR NAS

We trained our NAS SwinIR (which we denote as SwiNASIR) using the same hyperparameters as the original SwinIR paper. We trained on the DIV2K dataset [9] and evaluated on the Set5 [55], Set14 [56], BSD100 [57], Urban100 [58], and Manga109 [59] test datasets. High-quality and low-quality image pairs are generated with a bicubic interpolation kernel. We trained for 500K iterations and set the batch size to 32. The learning rate is initialized to 2e-4 and halved at iterations: [250K, 400K, 450K, 475K]. We used the Adam [15] optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

From Table II, we can see that the SwiNASIR and SwiNASIR+ model architectures built with NAS slightly outperform the manually defined models SwinIR and SwinIR+, respectively. We believe this is strong evidence for the correctness of SCaMP's NAS/HPO meta-parallelism strategies.

VI. Related Work

In this sections, we briefly review past research on image super-resolution (SR) and deep learning search strategies such as neural architecture search (NAS) and hyperparameter optimization (HPO) for its relevancy in our work.

Most prior works on training parallel image SR models focus on data-parallel CNNs [60], [61] or ViTs [52]. While these works focus on either the use of HPC systems in training large state-of-the-art models, or the tuning of HPC middleware for modern SR models, our work uses image SR as an example use case for meta-training paradigms (e.g. NAS and HPO). Specifically, we are interested in parallelism schemes *among* DL training settings, not *within* a single setup. Therefore, these prior works are complementary to ours.

Much work has been done on DL parallelism schemes. Both data-parallel [26], [62], [25] and model-parallel [20], [16], [17], [24], [63] designs and frameworks have emerged to tackle the problems of training in-core and out-of-core models, respectively. Similar to prior image SR model papers, these are complimentary to our work. Since DeepSpeed is a current frontrunner 3D parallism DL framework, we have chosen it to power ScaMP's training engine.

To our knowledge, we are the first to train a state-of-the-art super-resolution model with NAS. We have built atop the

TABLE II
QUANTITATIVE COMPARISON BETWEEN STANDARD SWINIR AND A MODEL TRAINED AUTOMATICALLY BY SCAMP'S HPO/NAS FEATURES. EVALUATIONS ARE CARRIED OUT FOR CLASSICAL IMAGE SR ON BENCHMARK DATASETS USING THE AVERAGE PSNR/SSIM AS AN EVALUATION METRIC. BEST AND SECOND BEST PERFORMANCE ARE COLORED RED AND BLUE, RESPECTIVELY.

Model	Scale	Training Dataset	Set5		Set14		BSD100		Urban100		Manga109	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
SwinIR	x2	DIV2K	38.35	0.962	34.14	0.9227	32.44	0.903	33.4	0.9393	39.6	0.9792
SwinIR+	x2	DIV2K	38.38	0.9621	34.24	0.9233	32.47	0.9032	33.51	0.9401	39.7	0.9794
SwiNASIR (Ours)	x2	DIV2K	38.42	0.9623	34.27	0.9236	32.48	0.9034	33.56	0.9406	39.7	0.9793
SwiNASIR+ (Ours)	x2	DIV2K	38.45	0.9625	34.32	0.9241	32.52	0.9035	33.65	0.9412	39.9	0.9795
SwinIR	x3	DIV2K	34.89	0.9318	30.77	0.8503	29.37	0.8124	29.29	0.8744	34.74	0.9518
SwinIR+	x3	DIV2K	34.95	0.9322	30.83	0.8511	29.41	0.813	29.42	0.8761	34.92	0.9526
SwiNASIR (Ours)	x3	DIV2K	35.09	0.9324	31.01	0.854	29.51	0.8153	29.65	0.879	35.03	0.9531
SwiNASIR+ (Ours)	x3	DIV2K	35.16	0.9328	31.07	0.8545	29.56	0.8159	29.75	0.8803	35.19	0.954
SwinIR	x4	DIV2K	32.72	0.9021	28.94	0.7914	27.83	0.7459	27.07	0.8164	31.67	0.9226
SwinIR+	x4	DIV2K	32.81	0.9029	29.02	0.7928	27.87	0.7466	27.21	0.8187	31.88	0.9423
SwiNASIR (Ours)	x4	DIV2K	32.9	0.9041	29.07	0.7945	27.9	0.7487	27.39	0.8231	31.99	0.9258
SwiNASIR+ (Ours)	x4	DIV2K	32.95	0.9046	29.717	0.796	27.93	0.7492	27.45	0.825	32.17	0.9269

work performed by previous works on NAS for ViT models, such as GliT [64] and Autoformer [65].

Attention-based methods have also been applied towards image super-resolution and were able to achieve state-of-the-art results [66], [67], [52]. TTSR [67] was one of the first transformers for image super-resolution. It uses attention mechanisms to efficiently transfer relevant textures from high-resolution reference images towards target low-resolution image. Proposed by Chen et al [66], IPT serves as a backbone model for various image restoration tasks based on large pre-trained Transformer. IPT depends on a huge number of parameters, large-scale datasets and contrastive learning to achieve multi-task compatibility. Recently, Swin Transformer [68] has laid path for more potential in image super resolution with its reduced computational complexity and multi-scale feature integration. SwinIR [52] is a image super-resolution transformer based on swin transformer and has been topping several benchmark with notable performance.

While NAS has been applied to find better CNN architectures [29], it is still hard to shift towards transformer architectures since search space of transformer models are extremely large and thus requires well-designed search space and aggressive search strategies. GLiT [64] designed a hierarchical neural architecture search method that aims to search for the optimal vision transformer from two levels with evolutionary algorithm. AutoFormer [65] proposed a new one-shot architecture search framework that applies weight entanglement to different blocks in the same layers to increase training efficiency on subnets.

Some prior works have studied parallelism for meta-learning tasks. In particular, the works in Tune [69]. As Hyperparameter Optimization (HPO) gradually becomes important in the model selection stage, more and more hyperparameter tuning frameworks emerge. Tune [69] was introduced as an open-sourced tuning framework that supports efficient distributed training on the Ray platform as well as a variety of state-of-the-art HPO algorithms. Optuna [70] features dynamic construction of search space, efficient pruning strategy and minimum setup requirements to deploy both small and large scale experiments. With a focus on production environment, Katib [71] was the first cloud native framework that elaborates on cross-team collaboration.

VII. CONCLUSION AND FUTURE WORK

State-of-the-art deep learning (DL) models are pushing the boundaries of existing fields while pioneering entirely new areas of study. However, choosing the DNN architecture settings or set of hyperparameters to maximize model accuracy requires a significant amount of trial-and-error and compute resources. In this paper, we present and evaluate SCaMP: a Scalable Meta-Parallelism framework for efficiently training multiple candidate architectures or hyperparameter sets concurrently. SCaMP supports flexible user configurations, efficient load balancing, and robust memory estimation. SCaMP is demonstrated on DL models such as Megatron [63] and SwinIR [52]. We report up to a 29% reduction in overall hyperparameter optimization time over basic data parallelism on 32 V100 GPUs on the Lassen HPC system, and a reduction in the proportion of overall time spent in communication from 28% to 15% on the ThetaGPU HPC system. To verify the correctness of our result, we have trained a state-of-the-art SwinIR model. We believe that the meta-parallelism strategies and load-balancing designs introduced in SCaMP will vastly reduce the time spent in HPO/NAS workloads.

As future work, we intend to extend SCaMP to support other DNN architectures beyond transformer-based models. The load-balancer currently only supports a single method of model parallelism, but we will add support for balancing multiple dimensions of model-parallelism (e.g. tensor parallelism within a node and pipeline parallelism across nodes) to efficiently train model-parallel subnets. Further, we will support and evaluate more NAS/HPO search strategies.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 05 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[2] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," 2022.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[4] 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

[5] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," 2018.

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

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[8] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[9] 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.

[10] "ILSVRC2012 Dataset," http://image-net.org/challenges/LSVRC/2012/index, 2012, [Online; accessed April-2016].

[11] "CIFAR-10 Dataset," 2019, [Online; accessed March 9, 2023]. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html

[12] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," *CoRR*, vol. abs/2101.00027, 2021. [Online]. Available: https://arxiv.org/abs/2101.00027

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[14] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," 2021. [Online]. Available: https://arxiv.org/abs/2110.02861

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[16] A. Jain, A. Shafi, Q. Anthony, P. Kousha, H. Subramoni, and D. K. Panda, "Hy-fi: Hybrid five-dimensional parallel dnn training onnbsp;high-performance gpu clusters," in *High Performance Computing: 37th International Conference, ISC High Performance 2022, Hamburg, Germany, May 29 – June 2, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 109–130. [Online]. Available: https://doi.org/10.1007/978-3-031-07312-0_6

[17] 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.

[18] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," 2020.

[19] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "Zero-offload: Democratizing billion-scale model training," 2021.

[20] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, *DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters*. New York, NY, USA: Association for Computing Machinery, 2020, p. 3505–3506. [Online]. Available: https://doi.org/10.1145/3394486.3406703

[21] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," 2020.

[22] 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.

[23] 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

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

[25] 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

[26] 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

[27] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," 2019. [Online]. Available: https://arxiv.org/abs/1905.01392

[28] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," *CoRR*, vol. abs/1707.07012, 2017. [Online]. Available: http://arxiv.org/abs/1707.07012

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2018. [Online]. Available: https://arxiv.org/abs/1802.01548

[30] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," 2020. [Online]. Available: https://arxiv.org/abs/2003.05689

[31] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," 2017. [Online]. Available: https://arxiv.org/abs/1707.05589

[32] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html

[33] R. J. G. van Sloun, O. Solomon, M. Bruce, Z. Z. Khaing, H. Wijkstra, Y. C. Eldar, and M. Mischi, "Super-resolution ultrasound localization microscopy through deep learning," 2018. [Online]. Available: https://arxiv.org/abs/1804.07661

[34] L. Zhang, H. Zhang, H. Shen, and P. Li, "A super-resolution reconstruction algorithm for surveillance images," *Signal Process.*, vol. 90, no. 3, p. 848–859, mar 2010. [Online]. Available: https://doi.org/10.1016/j.sigpro.2009.09.002

[35] P. Rasti, T. Uiboupin, S. Escalera, and G. Anbarjafari, "Convolutional neural network super resolution for face recognition in surveillance monitoring," in *Articulated Motion and Deformable Objects*, 2016.

[36] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[37] C. E. Duchon, "Lanczos filtering in one and two dimensions," *Journal of Applied Meteorology and Climatology*, vol. 18, no. 8, pp. 1016 – 1022, 1979.

[38] K. I. Kim and Y. Kwon, "Single-image super-resolution using sparse regression and natural image prior," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 6, pp. 1127–1133, 2010.

[39] Z. Xiong, X. Sun, and F. Wu, "Robust web image/video super-resolution," *IEEE Transactions on Image Processing*, vol. 19, no. 8, pp. 2017–2028, 2010.

[40] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," 2015. [Online]. Available: https://arxiv.org/abs/1501.00092

[41] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.04587

[42] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, "Residual dense network for image super-resolution," 2018. [Online]. Available: https://arxiv.org/abs/1802.08797

[43] J. Zhang, F. Zhan, Y. Yu, R. Wu, X. Zhang, and S. Lu, "Latent multi-relation reasoning for gan-prior based image super-resolution," 2022. [Online]. Available: https://arxiv.org/abs/2208.02861

[44] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," 2017. [Online]. Available: https://arxiv.org/abs/1707.02921

[45] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," 2016. [Online]. Available: https://arxiv.org/abs/1609.04802

[46] V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," 2022. [Online]. Available: https://arxiv.org/abs/2205.05198

[47] Q. Anthony, A. Awan, J. Rasley, Y. He, S. Aamir, M. Abduljabbar, H. Subramoni, and D. Panda, "Mcr-dl: Mix-and-match communication runtime for deep learning," *(Accepted, to be presented) 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2023.

[48] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-efficient distributed deep learning: A comprehensive survey," 2020. [Online]. Available: https://arxiv.org/abs/2003.06307

[49] Argonne National Laboratory, "Theta/ThetaGPU Machine Overview," https://www.alcf.anl.gov/support-center/theta/theta-thetagpu-overview, 2021, Accessed: March 9, 2023.

[50] H. Meur, E. Strohmaier, J. Dongarra, and H. Simon, "TOP 500 Supercomputer Sites," http://www.top500.org, 1993, [Online; accessed March 9, 2023].

[51] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, https://mvapich.cse.ohio-state.edu/, 2001, [Online; accessed March 9, 2023].

[52] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte, "Swinir: Image restoration using swin transformer," 2021. [Online]. Available: https://arxiv.org/abs/2108.10257

[53] "ViT-PyTorch," https://github.com/lucidrains/vit-pytorch, 2020, [Online; accessed February-2023].

[54] "Megatron-DeepSpeed," https://github.com/microsoft/Megatron-DeepSpeed, 2019, [Online; accessed February-2023].

[55] M. Bevilacqua, A. Roumy, C. Guillemot, and M. line Alberi Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *Proceedings of the British Machine Vision Conference*. BMVA Press, 2012, pp. 135.1–135.10.

[56] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *Curves and Surfaces*, J.-D. Boissonnat, P. Chenin, A. Cohen, C. Gout, T. Lyche, M.-L. Mazure, and L. Schumaker, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 711–730.

[57] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, 2001, pp. 416–423 vol.2.

[58] J.-B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5197–5206.

[59] Y. Matsui, K. Ito, Y. Aramaki, A. Fujimoto, T. Ogawa, T. Yamasaki, and K. Aizawa, "Sketch-based manga retrieval using manga109 dataset," *Multimedia Tools and Applications*, vol. 76, no. 20, pp. 21 811–21 838, nov 2016. [Online]. Available: https://doi.org/10.1007%2Fs11042-016-4020-z

[60] 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

[61] Q. Anthony, L. Xu, H. Subramoni, and D. K. D. Panda, "Scaling single-image super-resolution training on modern hpc clusters: Early experiences," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 923–932.

[62] 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

[63] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *CoRR*, vol. abs/1909.08053, 2019. [Online]. Available: http://arxiv.org/abs/1909.08053

[64] B. Chen, P. Li, C. Li, B. Li, L. Bai, C. Lin, M. Sun, J. yan, and W. Ouyang, "Glit: Neural architecture search for global and local image transformer," 2021. [Online]. Available: https://arxiv.org/abs/2107.02960

[65] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," 2021. [Online]. Available: https://arxiv.org/abs/2107.00651

[66] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao, "Pre-trained image processing transformer," 2020. [Online]. Available: https://arxiv.org/abs/2012.00364

[67] F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo, "Learning texture transformer network for image super-resolution," 2020. [Online]. Available: https://arxiv.org/abs/2006.04139

[68] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021. [Online]. Available: https://arxiv.org/abs/2103.14030

[69] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," 2018. [Online]. Available: https://arxiv.org/abs/1807.05118

[70] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019. [Online]. Available: https://arxiv.org/abs/1907.10902

[71] J. George, C. Gao, R. Liu, H. G. Liu, Y. Tang, R. Pydipaty, and A. K. Saha, "A scalable and cloud-native hyperparameter tuning system," 2020.