2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 978-1-6654-4066-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/IPDPS49936.2021.00071

SUPER: SUb-Graph Parallelism for TransformERs

Arpan Jain* †, Tim Moon‡, Tom Benson†, Hari Subramoni*, Sam Adé Jacobs†,
Dhabaleswar K Panda*, Brian Van Essen†

* *Department of Computer Science and Engineering*
The Ohio State University
Columbus, Ohio, USA
{jain.575, subramoni.1}@osu.edu,
panda@cse.ohio-state.edu

† *Center for Applied Scientific Computing*
‡ *Global Security Computing Applications Division*
Lawrence Livermore National Laboratory,
Livermore, California, USA
{moon13, benson31, jacobs32, vanessen1}@llnl.gov

***Abstract*—Transformer models have revolutionized the field of Natural Language Processing (NLP) and they achieve state-of-the-art performance in applications like machine translation, question answering, regression, and summarization. However, training Transformers is challenging because of their large memory and compute requirements. The literature contains several approaches to parallelize training, like layer parallelism and pipeline parallelism, but they are optimized to benefit out-of-core models and they don't exploit the inherent parallelism in Transformer models. Other work uses model parallelism to achieve weak scaling by increasing the model size. In this paper, we propose sub-graph parallelism that provides a significant performance improvement over pure data parallelism with a fixed number of resources, and as an additional technique for strong- and weak-scaling without increasing model capacity. Our technique accelerates the training of Transformer models and we generalize the concept to any neural network with multiple branches. We optimize the communication for sub-graph parallelism and combine it with data parallelism to scale performance up to 1024 GPUs. To decrease communication overheads, we propose a topology-aware scheme that limits inter-node communication. Finally, we empirically compare sub-graph parallelism with pure data parallelism and demonstrate its performance benefits in end-to-end training.**

***Index Terms*—scalable deep learning, transformers, sub-graph parallelism, algorithms, deep learning, distributed training**

I. Introduction

A. Motivation

The task of training Transformer models is one of the most important workloads in contemporary deep learning. Since their development in 2017 [1], Transformers have continually pushed the state of the art in natural language processing [2]–[4] and they have recently achieved impressive results in computer vision and object recognition [5]–[7]. The distinguishing feature of a Transformer is the attention operation, which allows the model to selectively focus on specific entries in a data sequence. Multi-head attention performs multiple attention operations independently, allowing for models that can learn complicated patterns like the grammars of natural languages. While there are many variants of the original model like BERT [8], GPT-2 [2], GPT-3 [4], and T5 [3], they share the same basic architecture (autoencoder built of stacked multi-head attention operations) and differ primarily in the number of trainable parameters, in whether they are encoder- or decoder-centric, and in the objective function for training.

While they achieve excellent results, Transformers have significant memory, compute, and data requirements to train. Increasing the number of trainable parameters tends to improve learning quality, so the best models approach or exceed the memory capacity of a single GPU. In addition, the large number of parameters necessitates very large training sets in order to avoid overfitting, making scalable performance a necessity to train a Transformer in a reasonable time.

Traditional data-parallel approaches are poorly suited to this task since they require redundant copies of the model on every parallel process. Even if a model fits on a GPU, the mini-batch sizes may be too small to achieve good compute efficiency. Memory pressure can be mitigated with out-of-core training, but that requires the complexity and overhead of fine-grained data transfers between CPUs and GPUs. Furthermore, the compute graph of a Transformer contains branching sections where each individual branch is relatively inexpensive, so executing the entire graph on each GPU will incur the latency cost of many kernel launches. Finally, the large size of Transformers means that synchronizing the parameters or parameter gradients will incur major communication costs, especially at scale. Work such as Megatron-LM [9] lays out the initial groundwork for sub-graph parallelism on Transformers, focusing on enabling weak-scaling by increasing the model capacity. In this work we present a generalization of sub-graph parallelism that applies to both strong- and weak-scaling, without requiring changes in the neural network architecture to increase parallelism.

B. Challenges for Sub-graph Parallelism

We propose a hybrid of data parallelism and sub-graph parallelism to efficienty train deep neural networks (DNNs) with multi-branch architectures. Such models, like Transformers and ResNeXt [10], present excellent opportunities for parallelization, namely by distributing the branches between parallel processes and processing them independently. Distributing sections of the compute graph between processes also distributes the memory and compute, alleviating some of the specific difficulties with training Transformers. However, this approach requires surmounting several challenges:

Challenge 1: Exploiting the inherent parallelism in multi-branch DNN architectures

Multi-branch DNN architectures usually have branches that are individually short and somewhat inexpensive. Therefore,

978-1-6654-4066-0/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPS49936.2021.00071

the communication overheads can easily overwhelm the gains from parallelization, resulting in a net loss in performance. Furthermore, the splits and joins at the beginning and end of branches cannot be neatly assigned to a specific branch. These layers are the logical point for communication, so it is important to optimize their performance to minimize communication overheads.

Challenge 2: Designing efficient communication patterns

Sub-graph parallelism requires a large volume of communication at the beginning and end of each branch. However, the branching sections are usually organized sequentially, so it is impractical to overlap computation and communication. In other words, sub-graph parallelism introduces blocking communication into the the forward and backward passes of training. The performance of training is thus highly sensitive to the efficiency of the communication patterns. In addition, there are multiple ways to split or join branches, each with different communication characteristics, so we must implement and optimize multiple designs.

Challenge 3: Maintaining performance at scale

Multi-branch architectures, especially Transformers, are computationally expensive and they require training on large data sets. Parallelization solutions that can scale to large numbers of GPUs are especially important to train in a reasonable time frame. However, increasing the number of GPUs involved in communication tends to increase communication overhead. Sub-graph parallelism involves blocking communication, so the scalability of sub-graph parallelism is highly sensitive to communication overhead.

C. Contributions

- A generalized hybrid of data and sub-graph parallelism (D&SP) that significantly outperforms pure data-parallelism, and strong- and weak-scales the training of multi-branch DNN architectures like Transformers.
- Topology-aware optimizations of communication patterns to achieve better scalability on a large number of GPUs.
- Performance evaluations on three Transformer models using a large GPU supercomputer. Compared with existing techniques, we observe that D&SP achieves up to a $3.05\times$ speedup with T5 and $1.5\times$ with GPT-2 when using the same amount of resources.
- A publicly available open-source implementation at `https://github.com/LLNL/lbann`.

II. Background

A. Transformer Architecture

The core operation in a Transformer model is multi-head attention [1]. Given L query vectors $q_i \in \mathbb{R}^{d_k}$ and S key-value vector pairs $(k_i, v_i) \in \mathbb{R}^{d_k} \times \mathbb{R}^{d_v}$, the basic attention operation compares the queries and keys to construct L weighted sums of the value vectors. More precisely, packing these column vectors into matrices $Q \in \mathbb{R}^{d_k \times L}$, $K \in \mathbb{R}^{d_k \times S}$, $V \in \mathbb{R}^{d_v \times S}$,

$$\text{Attention}(Q, K, V) = V \,\text{softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right) \qquad (1)$$

Softmax is applied independently to each matrix column. A masking matrix $M \in \{0, -\infty\}^{S \times L}$ is sometimes applied to the softmax input and dropout to the softmax output. Multi-head attention (see Figure 5a) performs h independent attention operations,

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= W^O \,\text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) \\ \text{with head}_i &= \text{Attention}\left(Q_i, K_i, V_i\right)\end{aligned} \qquad (2)$$

We have $Q_i = \text{Slice}_i\left(W^Q Q\right)$, $K_i = \text{Slice}_i\left(W^K K\right)$, $V_i = \text{Slice}_i\left(W^V V\right)$ and the slice and concatenation operations are across matrix columns, i.e. unstacking and stacking respectively. $W^Q \in \mathbb{R}^{hd_k \times d_{\text{model}}}$, $W^K \in \mathbb{R}^{hd_k \times d_{\text{model}}}$, $W^V \in \mathbb{R}^{hd_v \times d_{\text{model}}}$, and $W^O \in \mathbb{R}^{d_{\text{model}} \times hd_v}$ are trainable matrix parameters. Self-attention involves passing identical Q, K, and V into multi-head attention.

Transformer models are typically built out of stacks of self-attention operations and multilayer perceptrons (MLPs). The MLPs usually have one large hidden layer and are applied separately to each sequence vector.

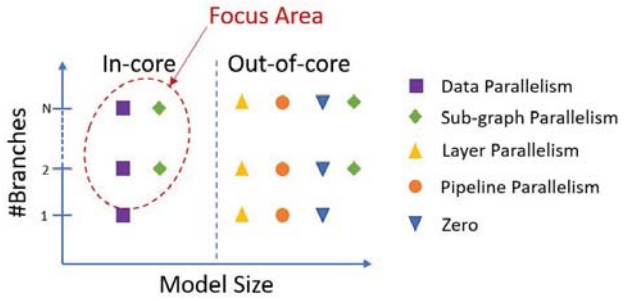


Fig. 1: Strategies for distributed training of Transformers. We focus on distributed algorithms to accelerate in-core models.

B. Related Work

The computational requirements for training DNNs have doubled every three to four months since 2012 [11], leading to the development of several strategies for distributed training. See Figure 1 for an overview of existing techniques. Data parallel training for convolutional neural networks [12] has proven to be enormously successful, leading to incredibly quick training times [13]. However, as discussed in Section I-A, this does not appear as promising for training Transformers due to their memory and compute requirements. Pipeline parallelism is a very general strategy that has been applied to achieve modest scaling on Transformers, but it requires the complexity of micro-batching and introduces the bubble overhead of filling and draining the pipeline [14]. It also has similar high-volume communication requirements as our approach, but is unable to apply our topology-aware optimizations to the communication patterns. [15] introduces a clever scheme to reduce memory pressure by distributing model parameters between processes, but it requires global collective communication in the forward pass as well as backward pass.

Other work has explored algorithmic changes to the Transformer model like introducing sparsity [16] or improving the asymptotics of the attention operation [17]. We note that most of the literature focuses on accelerating out-of-core models, i.e. models that cannot fit inside the memory of a single GPU. Our approach does not have the same memory requirements as data parallelism, but we focus primarily on optimizing in-core models and use data parallelism as a performance baseline. That said, our technique is flexible and may be composed with the other approaches. MetaFlow [18] and TASO [19] proposed compiler-level optimizations to generate efficient graph substitutions automatically; however, they executed the generated graph on a single GPU and did not take advantage of distributed processing.

As discussed earlier, we build upon and generalize some of the work presented with Megatron-LM [9]. They implement a simple form of sub-graph parallelism combined with layer parallelism, enabling training of larger models by increasing the model capacity. The result is similar to our D&SP-cSub strategy. However, they do not address the challenges of data movement nor the opportunities for topology-aware optimizations. Furthermore, since they use the same local mini-batch throughout the entire model, within and without the branching sections, they limit the memory scaling in the non-branching sections. And finally, they rely on achieving a specific sequence of splits in their data tensors to perform communication. Our work seeks to generalize these ideas and offer more flexibility for future parallelism methods.

We remark that this work is a generalization of the SUMMA algorithm for distributed matrix multiplication [20]. If we decompose a fully-connected layer by slicing the input tensor, applying pieces of the weights matrix to each slice, and adding the results together, applying our D&SP scheme essentially recovers SUMMA. Decomposing a convolutional layer in a similar manner, possibly with multiple levels of sub-graph parallelism, recovers channel and filter parallelism [21].

III. Design of Data and Sub-Graph Parallelism (D&SP)

We investigate and propose several designs to exploit the inherent parallelism in Transformer models. Throughout this section we discuss how computations are parallelized across multiple MPI processes in a distributed training framework (LBANN), with one GPU per process.

A. Sub-graph Parallelism

Transformer models are examples of multi-branch DNN architectures. These architectures have compute graphs similar to Figure 2, where a "common layer" splits into multiple "branch layers", which eventually join into another common layer. The computation within a branch can be arbitrarily complex, and can be accelerated as long as the branches are independent and have similar compute requirements. For Transformers, each head in multi-head attention can be considered as a branch, with splits at the slice layers and joins at the concatenation layers.

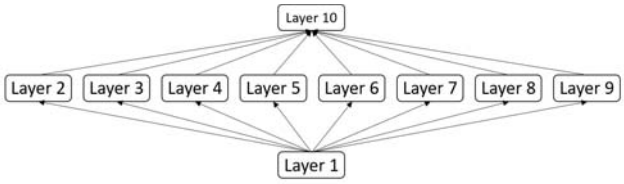


Fig. 2: Simple example of a multi-branch DNN architecture. A common layer (Layer 1) splits into eight branch layers (Layers 2-9), which join back into another common layer (Layer 10).

In pure sub-graph parallelism, the branches in the compute graph are divided into sub-graphs and each parallel process executes one sub-graph. Figure 3 shows an example of 4-way sub-graph parallelism. Observe that the common layers (layers 1 and 10) are all executed on GPU 1, and thus experience no benefit from parallelization. If the common layers involve significant computation, this results in under-utilization and load imbalance issues. In addition, the maximum number of processes is one per branch. If individual branches are sufficiently inexpensive, there may also be a point where the extra compute from adding processes is offset by the growth in communication overhead. The poor parallel efficiency and the limits to scaling compel us to refine this design.

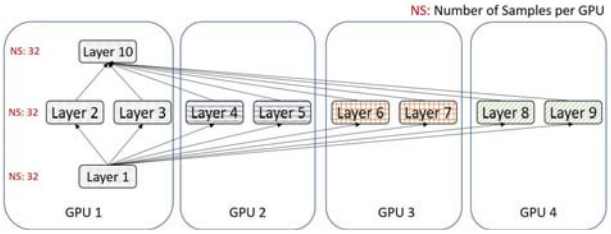


Fig. 3: 4-way sub-graph parallelism for the multi-branch architecture in Figure 2.

B. Data and Sub-graph Parallelism (D&SP)

The issues in Section III-A can be addressed by combining data parallelism with sub-graph parallelism. Pure data parallelism involves distributing the data samples in a training step mini-batch between processes. Each process applies a duplicate copy of the model to its local data and accumulates parameter gradients between processes. This approach can scale indefinitely if the mini-batch size is made sufficiently large and the communication cost of gradient accumulation is small. However, the redundant model copies on each process impose high memory requirements. If the model is large, then it may not fit on a single process or the per-process mini-batch size might be too small to get good compute efficiency. Also, the scalability in practice is limited since the communication overhead grows with the number of processes.

We propose Data and Sub-graph Parallelism (D&SP) as a strategy to exploit the inherent parallelism in multi-branch DNN architectures while retaining the scalability properties of data parallelism. Common layers are parallelized in a standard parallel fashion and branch layers are divided into

sub-graphs as discussed in Section III-A. However, sub-graphs are not executed by a single process, but by a "sub-grid" of multiple processes. Each sub-grid operates independently and uses data parallelism internally. Figure 4 shows 4-way D&SP where each sub-grid consists of one GPU. Note the all-to-all communication required at the split and join layers (layers 1 and 10). If the global mini-batch size is 32, then the per-process mini-batch size is 8 in the common layers and 32 in the branch layers.

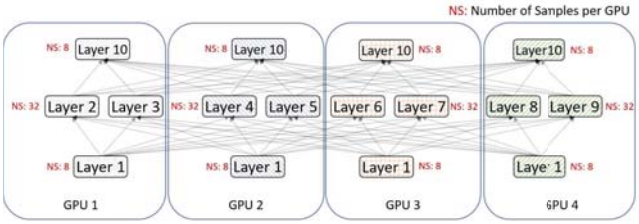


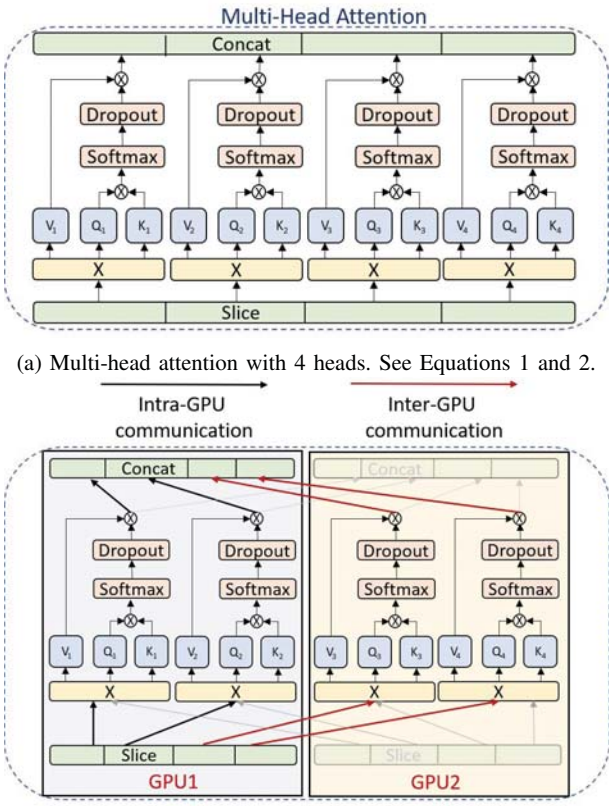
Fig. 4: 4-way D&SP with 4 processes for the multi-branch architecture in Figure 2. Branch layers use 4-way sub-graph parallelism and common layers are distributed using data parallelism.

Unlike pure sub-graph parallelism, D&SP manages to parallelize the common layers, achieves good load balancing, and can accommodate a number of processes up to the mini-batch size. Within branching sections, it improves on data parallelism by reducing the memory requirements for the model parameters and reducing the number of processes that participate in gradient accumulation. In addition, processes have larger local mini-batch sizes in the branching sections than with data parallelism and perform fewer operations, so they tend to achieve higher compute efficiency on GPUs. The major challenge is optimizing the all-to-all communication at the beginning and end of branching sections.

To make this approach concrete, consider applying D&SP to multi-head attention as shown in Figure 5. For a global mini-batch size of 32, the per-process mini-batch size is 16 in the common layers. In the first process, the slice layer will divide its 16 input sequences into four pieces, perform intra-GPU transfers with the first two pieces, and send the second two pieces to the second GPU. Each of its two heads will also receive 16 input sequence pieces from the second process, resulting in a local mini-batch size of 32. An opposite communication pattern occurs at the final concatenation layer.

C. D&SP-cSub

Pure sub-graph parallelism and D&SP lie at two extremes of a spectrum: sub-graph parallelism assigns common layers to a single process and D&SP distributes common layers between all available processes. However, the optimal number of processes may differ between common layers and branch layers. In particular, the communication cost of gradient accumulation increases with the number of processes, so placing common layers on a subset of available processes may give better performance. This is more likely if the mini-batch size is small and poor compute efficiency eliminates the performance



(a) Multi-head attention with 4 heads. See Equations 1 and 2.

(b) 2-way D&SP with 2 processes for multi-head attention with 4 heads. Compute and communication patterns are highlighted for GPU1.

Fig. 5: D&SP for multi-head attention.

benefit of running with extra processes. D&SP-cSub is a variant of D&SP that places the common layers on one of the sub-grids. Instead of all-to-all communication patterns at the beginnings and ends of branching sections, splits will involve one-to-all communication and joins will involve all-to-one communication. Note that D&SP and D&SP-cSub will be better in different situations since their performance depends on factors like the mini-batch size, the model size, and the ratio between compute and communication.

IV. Enhancing Communication Patterns in D&SP and D&SP-cSub

The performance of sub-graph parallelism depends heavily on its communication patterns. Branch layers cannot start their computation without receiving data from the preceding common layers, so communication is a blocking operation. Since we can not hide communication behind computation, we must take care to minimize communication overheads. We observe that the following common layers can split into branching sections:

- **Split**: Copy the input tensor and send to each branch.
- **Slice**: Slice the input tensor along a given dimension and send each piece to its own branch. For our case, we slice

tensors across the inner-most dimension (the embedding vector dimension) into equally-sized pieces.

The following common layers can join branching sections:

- **Sum**: Add input tensors together.
- **Concatenation**: Concatenate the input tensors along a given dimension. For our case, we concatenate along the inner-most dimension.

Observe that the split and sum layers are closely related: the forward and backward passes of the split layer are equivalent to the backward and forward passes of the sum layer, respectively. The slice and concatenation layers are similarly related. With these insights, we first develop a general communication pattern before optimizing special cases.

A. General Communication Design

We implemented sub-graph parallelism within the LBANN deep learning toolkit [22]. Distributed tensors are managed with Hydrogen [23], a GPU-enabled fork of the Elemental library for distributed linear algebra [24]. LBANN implements data parallelism by storing activations and error signals in 2D distributed matrices where each column corresponds to a mini-batch sample. A Hydrogen distributed matrix is distributed over a "grid", which is a group of parallel processes with one GPU per process. For our purposes, a grid is equivalent to an MPI communicator. The matrices have column-major ordering and matrix columns are distributed over the grid in a round-robin manner, i.e. the [STAR,VC] distribution in Elemental's notation, as shown in Figure 6.

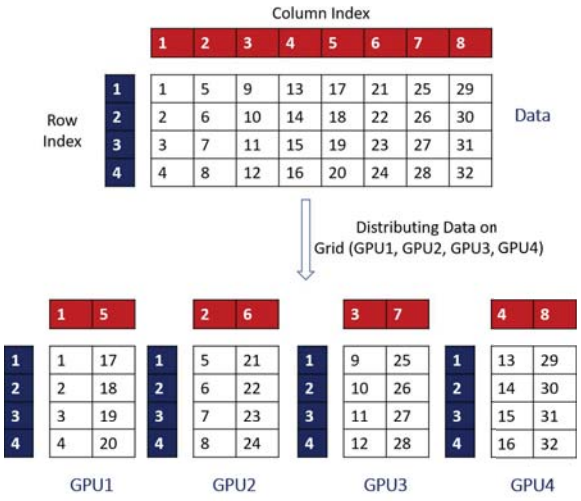


Fig. 6: [STAR,VC] distribution of a 2D matrix over four parallel processes. The matrix columns are distributed over the processes in a round-robin manner.

Sub-graph parallelism simply involves distributing the data matrices of common layers on a common grid and the data matrices of branch layers on smaller sub-graph grids. The common grid is the global grid for D&SP and the first sub-graph grid for D&SP-cSub. The major challenge lies in optimizing the communication between the common grid and sub-graph grids whenever branchs are split or joined. We began by optimizing Hydrogen's `TranslateBetweenGrids` function to support transferring [STAR,VC] matrices between any two grids. It involves an all-to-all communication pattern, as shown in Figure 7, and is built out of point-to-point communication primitives instead of collectives since the number of send and receive processes may differ and the message sizes may not follow a nice pattern. In addition, this approach is not scalable with the number of sub-graphs grids since it requires one call of `TranslateBetweenGrids` per sub-graph grid. Thus, having a general fallback for communication between grids, we move on to optimize some important special cases.

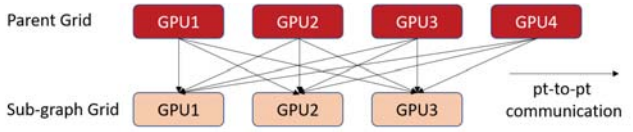


Fig. 7: All-to-all communication pattern for Hydrogen `TranslateBetweenGrids` function.

B. Split and Sum Layers

Split and sum layers have the simplest communication pattern for sub-graph parallelism. A naive implementation of the split layer forward pass would call the `TranslateBetweenGrids` function once per sub-graph grid. Let us assume from now on that the sub-graph grids have equal size, are disjoint, and divide the global grid. Since LBANN data matrices distribute columns in a round-robin manner, process i in the common grid will own columns corresponding to process $i \mod n_g$ in each sub-graph grid, where n_g is the sub-graph grid size, as shown in Figure 8. Thus, if we form n_g disjoint sub-communicators, each corresponding to a process in the sub-graph grids, all the necessary communication takes place within these sub-communicators. For D&SP, a gather collective on each sub-communicator will deliver the correct matrix columns from the common grid to the sub-graph grids (with some column reshuffling required afterwards). Note that these simultaneous collectives on disjoint sub-communicators can be considered as a single *segmented* collective.

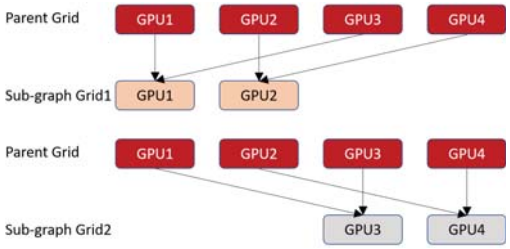
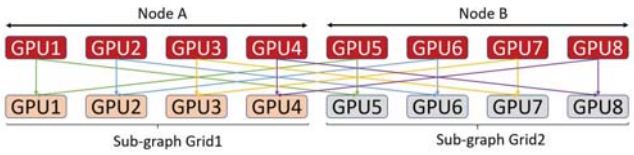
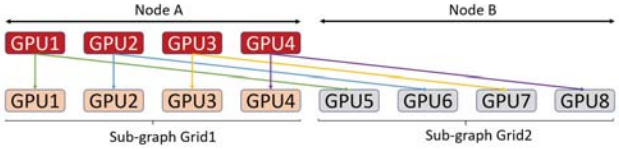


Fig. 8: Communication pattern for the forward pass of the split layer, using D&SP with `TranslateBetweenGrids`. Observe the all-to-all pattern within the two disjoint sub-communicators.

We can do even better by consolidating all of the `TranslateBetweenGrids` calls into a single segmented collective. As shown in Figure 9, the processes in the common grid are sending the same data to each sub-graph grid, so the repeated segmented gathers in D&SP can be replaced with a segmented all-gather. For D&SP-cSub, the communication is a segmented broadcast. Applying similar analysis to the sum layer forward pass finds that D&SP involves a segmented reduce-scatter and D&SP-cSub a segmented reduce.



(a) D&SP



(b) D&SP-cSub

Fig. 9: Communication patterns for the forward pass of the split and slice layers, using D&SP and D&SP-cSub.

C. Slice and Concatenation Layers

As discussed in Sections II-A and III-B and shown in Figure 5b, slice and concatenation layers are required to implement D&SP for multi-head attention. We are interested particularly in slicing and concatenating along the inner-most dimension of 3D data tensors. Let us consider the forward pass of the slice layer for now and assume that the sub-graph grids evenly divide the global grid. Recall that LBANN data matrices interpret the outer-most tensor dimension (mini-batch size) as the matrix width and flatten the remaining dimensions into the matrix columns. Conveniently, we can use basic matrix operations to reorder the data into a format amenable to communication. As shown in Figure 10, it simply requires resizing the data matrix and transposing. Resizing is a logical operation that does not require any data movement and the resize dimensions determine the slice dimension. The columns of the resulting matrix are transferred and then reordered with another transpose and resize. The communication pattern is similar to the split layer's (see Section IV-B and Figure 9), except that each sub-graph grid receives different data from the common grid. For D&SP, this means that the segmented all-gather should logically be replaced with a segmented all-to-all. However, we use a GPU communication library, NCCL, without an optimized all-to-all implementation. We find that is faster in practice to perform an all-gather and to discard unneeded data than to implement all-to-all out of point-to-point primitives. For D&SP-cSub, the split layer's segmented broadcast is replaced with a segmented scatter. A similar approach can be applied to the forward pass of the concatenation layer: D&SP involves a segmented all-to-all (in practice an all-gather) and D&SP-cSub a segmented gather.

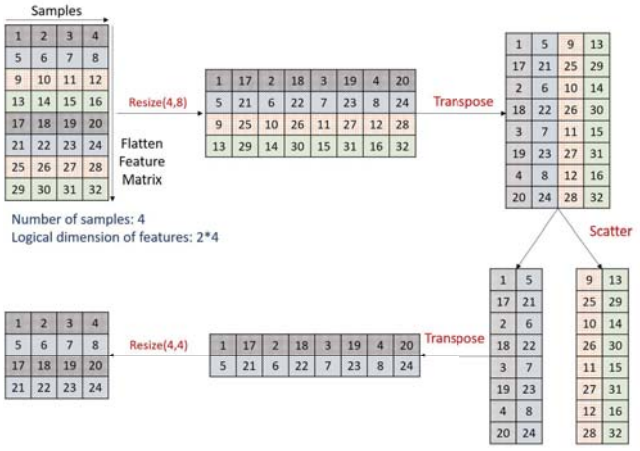


Fig. 10: Implementation of slice layer with D&SP-cSub. Note that the matrix is column-major and that only basic matrix operations are required.

V. Topology-aware D&SP and D&SP-cSub

D&SP and D&SP-cSub can be scaled to a large number of GPUs by increasing the number or size of sub-graph grids. This involves two phases of communication: segmented collectives between sub-grids at branch splits and joins, and segmented all-reduces within sub-grids to accumulate parameter gradients. The gradient all-reduces can be overlapped with compute, in a similar manner as standard data parallel training, so we focus on the communication at branch splits and joins since it is blocking and typically involves large data volumes. Recall that the sub-communicator size in the segmented collectives is equal to the number of sub-grids. Thus, assuming a perfectly uniform network topology, keeping a fixed number of sub-grids and scaling the size of the sub-grids should have no impact on the communication overhead. This would just increase the number of disjoint sub-communicators operating in parallel.

However, real systems are not so homogeneous. GPUs are resident within compute nodes, and communication within a node has much better latency and bandwidth than communication between nodes. Intra-node communication is often optimized by vendors in both software and hardware, e.g. with CUDA peer-to-peer communication primitives and NVLink. Communication costs between nodes can also vary based on the network topology, e.g. if the nodes are resident on different server racks. Thus, even if a communication pattern involves the same number of GPUs, the run time can vary significantly based on process placement.

We can model the communication cost of D&SP by counting the number of messages required when splitting branches (see Section IV-B),

$$\text{Messages} = \text{IntraNode} + \text{InterNode} \tag{3}$$

Let us say g is the number of sub-grids, n the number of processes, n_{node} the number of processes per node, and $n_g = n/g$ the sub-grid size. Observe that the segmented all-to-all communication pattern implies Messages $= ng$. With the naive communication pattern shown in Figure 9a,

$$\text{IntraNode}_{\text{naive}} = \begin{cases} n & n_g \leq n_{\text{node}} \\ n_{\text{node}} g & n_g > n_{\text{node}} \end{cases} \quad (4)$$

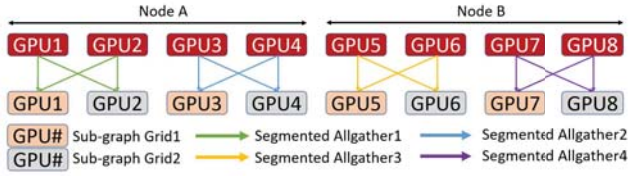
$$\text{InterNode}_{\text{naive}} = \begin{cases} n\,(g-1) & n_g \leq n_{\text{node}} \\ (n - n_{\text{node}})\,g & n_g > n_{\text{node}} \end{cases} \quad (5)$$

We propose a simple topology-aware optimization to minimize the communication volume between nodes. If we reorder the processes in the global grid so that they are distributed round-robin over the compute nodes, as shown in Figure 11 then we maximize the intra-node communication volume. In fact, the segmented collectives have no inter-node communication at all if the sub-grid size is less than the number of GPUs per node. Equations 4 and 5 become

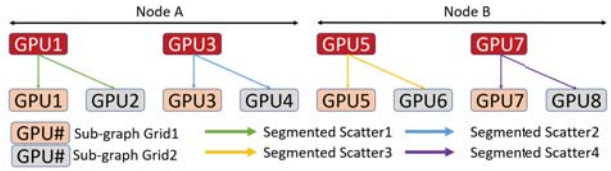
$$\text{IntraNode}_{\text{topo}} = \begin{cases} ng & g \leq n_{\text{node}} \\ n n_{\text{node}} & g > n_{\text{node}} \end{cases} \quad (6)$$

$$\text{InterNode}_{\text{topo}} = \begin{cases} 0 & g \leq n_{\text{node}} \\ n\,(g - n_{\text{node}}) & g > n_{\text{node}} \end{cases} \quad (7)$$

Thus, this modification to D&SP and D&SP-cSub significantly reduces the communication overhead of segmented collectives and reduces its sensitivity to the network topology at scale. Although we haven't implemented it, we remark that the inter-node communication could be optimized in a similar manner by reordering nodes round-robin over server racks.



(a) D&SP



(b) D&SP-cSub

Fig. 11: Communication patterns for the forward pass of the split and slice layers, using topology-aware D&SP and D&SP-cSub. Compare with Figure 9 and observe that there no inter-node communication since the number of sub-graph grids is less than the number of GPUs per node.

VI. Evaluation

A. Setup

a) Hardware: We conducted all our experiments on the Lassen supercomputer [25] at Lawrence Livermore National Laboratory. A compute node consists of two 22-core IBM Power9 CPUs and four Nvidia V100 GPUs. The GPUs have 16 GB HBM2 memory and each CPU has NVLink interconnects with two GPUs. Racks are composed of 18 nodes connected with InfiniBand EDR.

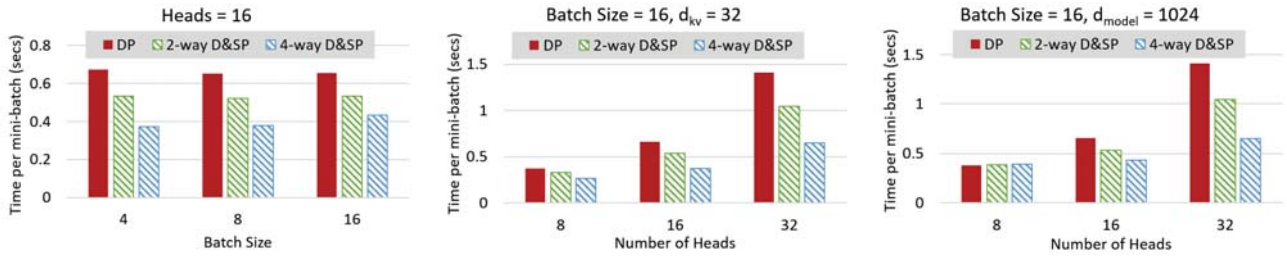
b) Deep Learning Framework: We implemented sub-graph parallel training in LBANN 0.100 [22]. LBANN was built with GCC 7.3.1, Spectrum MPI 10.3.1, CUDA 10.1, cuDNN 7.6.4, NCCL 2.4.2, Aluminum 0.5.0 [26], Hydrogen 1.5.0 [24].

B. Evaluation Setup and Performance Metrics

We perform experiments on several Transformer models using DP (pure data parallelism), D&SP (Section III-B), and D&SP-cSub (Section III-C). The models are trained with samples from the WMT14 English-German data set [27]. We focus on performance results since our algorithm is mathematically equivalent to single-GPU training and data-parallel training. Since we are particularly interested in steady-state performance, we perform a few warmup iterations (32 iterations) and report median mini-batch times (128 iterations). We perform both weak-scaling studies (keeping the per-GPU mini-batch constant while increasing the number of GPUs) and a strong-scaling study (keeping the global mini-batch constant while increasing the number of GPUs).

C. Single Node Evaluation with the Original Transformer

We performed single-node experiments with the original "Attention Is All You Need" Transformer model [1] and present them in Figure 12. Figure 12a shows the effect of varying the mini-batch size on the mini-batch time. With a mini-batch size of 4, 4-way D&SP achieved a $1.8\times$ speedup relative to pure DP. Note the mini-batch time with DP was independent of mini-batch size, even when increasing the mini-batch size increases the compute requirements, indicating that its performance was dominated by kernel launch overheads rather than compute. On the other hand, the run time of D&SP increased slightly with the mini-batch size. Figures 12b and 12c show the effect of increasing the number of heads in each multi-head attention operation. When the compute per head was kept constant (Figure 12b), the run time roughly scaled with the number of heads (as expected) and the benefit of D&SP increased to achieve a $2.16\times$ speedup at 32 heads. Figure 12c shows an increase in run time when keeping the total compute constant, indicating compute inefficiency due to over decomposition. While the run time of DP grew proportionally with the number of heads, implying a run time completely dominated by kernel launch overheads, it was sub-linear with D&SP. All these results demonstrate that D&SP can achieve moderate compute efficiency in situations where DP does not, e.g. when the mini-batch size is small or the number of heads is large.



(a) Different global mini-batch (b) Constant compute per head (c) Constant total compute

Fig. 12: Single-node performance of the original Transformer model, varying the global mini-batch size and the number of heads h in multi-head attention. Recall the d_k, d_v, and d_{model} parameters for multi-head attention from Section II-A. In these experiments, $d_{kv} = d_k = d_v$ determines the amount of compute per head and $d_{\text{model}} = hd_{kv}$ determines the total compute.

D. Weak Scaling with GPT-2

Figure 13 shows a weak-scaling study with GPT-2 [2]. On 1 node (4 GPUs), 4-way D&SP achieved a $1.47\times$ speedup relative to DP. However, we see that the advantage disappeared with larger numbers of compute nodes. Our profiling indicates that the primary limit to scaling for D&SP was the segmented all-reduces. Disabling the gradient all-reduces in the backward pass (without affecting any other computation or communication) resulted in perfect weak scaling. This result provides a compelling use-case for GPU communication packages to optimize the performance of segmented collectives, i.e. parallel collectives on disjoint sub-communicators. This would be especially interesting for the case where the number of sub-graph grids matches the number of GPUs per node, since communication between sub-grids (e.g. redistributing between parent and sub-graph grids) would be purely intra-node and communication within a sub-grid (i.e. gradient accumulation) would be purely inter-node.

Observe that D&SP and D&SP-cSub achieved nearly identical performance when the number of nodes was small, suggesting that the common layers have insufficient compute efficiency to benefit from increased parallelism. However, D&SP-cSub showed better scaling behavior, which can be attributed to the smaller grid size for gradient all-reduces in the common layers.

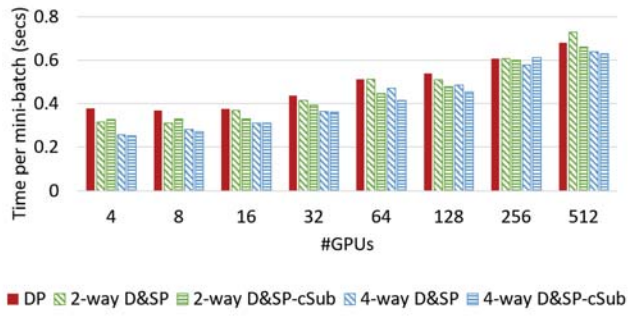
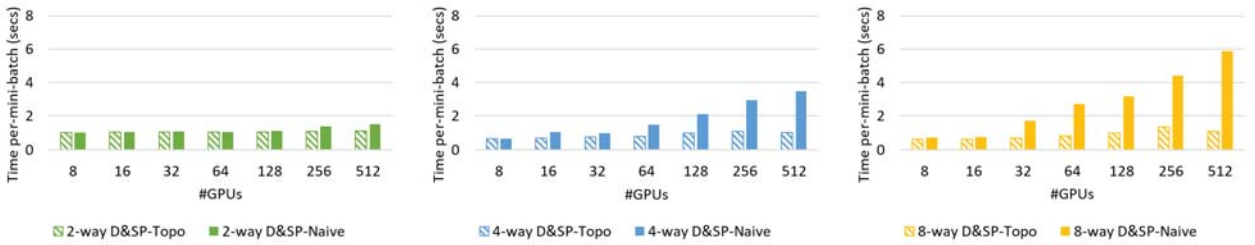


Fig. 13: Weak-scaling study with GPT-2 model. The per-GPU mini-batch size was 1 and all of the D&SP runs used topology-aware communication.

E. Weak Scaling with T5

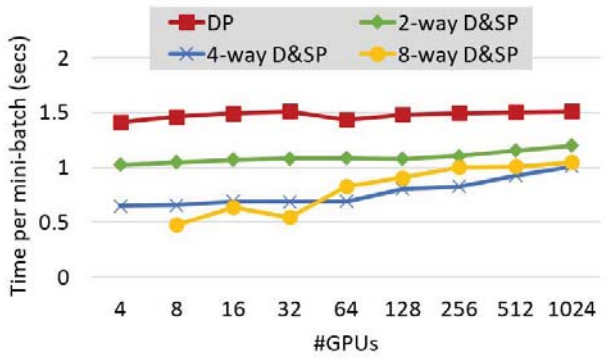
Our final weak scaling evaluation was with a T5 model [3]. In particular, we trained with a variant of T5-Large with 32 heads (T5-Large-Mod). This model is substantially more compute-intensive than GPT-2, to the extent that gradient all-reduces can be almost fully overlapped with compute. The weak-scaling study in Figure 14 demonstrates the importance of topology-aware communication. The naive communication pattern did not weak-scale due to the high volume of inter-node communication, but topology-aware communication achieved near-perfect weak-scaling.

Figure 15 shows a weak scaling study with DP and topology-aware D&SP. We were not able to evaluate D&SP-cSub because the common layers were too big to store in a single sub-graph grid. DP, 2-way D&SP, and 4-way D&SP all showed excellent weak-scaling. On 4 GPUs, 4-way D&SP was up to $2.22\times$ faster than DP with a per-GPU mini-batch size of 1 and up to $2.17\times$ faster with a per-GPU mini-batch size of 4. 8-way D&SP was the outlier since it involved high-volume inter-node communication. With a per-GPU mini-batch size of 1, the extra communication cost was minor and more than offset by the improved compute efficiency, resulting in a $3.05\times$ speedup on 8 GPUs relative to DP. However, increasing the per-GPU mini-batch size to 4 improved the compute efficiency of 2-way and 4-way D&SP, reducing the relative efficiency gains of 8-way D&SP. In addition, the extra communication cost of inter-node communication in 8-way D&SP became more onerous with a greater volume of data. The result was that 8-way D&SP had about the same performance as 4-way D&SP. Also observe that 8-way D&SP experienced more fluctuations in performance than 2-way and 4-way D&SP. In part this may be attributed to node placement across multiple racks. If so, this could be ameliorated with multi-level topology-awareness to reduce the amount of communication between racks, but that would greatly increase the algorithm's complexity. For now, we conclude that using more sub-grids than there are GPUs per node is a complicated decision that depends on the model characteristics, mini-batch size, and network topology.

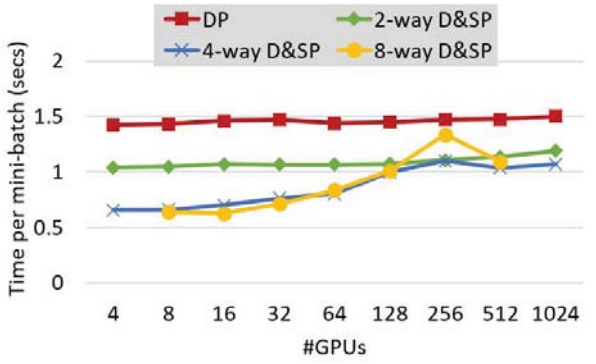


(a) 2-way D&SP (b) 4-way D&SP (c) 8-way D&SP

Fig. 14: Weak-scaling study with T5-Large-Mod model, comparing D&SP with naive and topology-aware communication. The per-GPU mini-batch size was 4.



(a) Per-GPU mini-batch size of 1



(b) Per-GPU mini-batch size of 4

Fig. 15: Weak-scaling study with T5-Large-Mod model. All of the D&SP runs used topology-aware communication.

F. Strong Scaling the "Attention Is All You Need" Transformer

Our implementation of D&SP is tuned to provide an algorithmic advantage over pure data parallelism, as well as the ability to apply strong-scaling when training with more compute resources. Figure 16 illustrates both of these improvements on the original "Attention Is All You Need" Transformer model. Peak strong-scaling is achieved using 16 GPUs on 4 compute nodes, where D&SP is able to achieve a $1.84\times$ speedup relative to the data-parallel baseline with 4 GPUs, a parallel efficiency of 46%. Scaling beyond that point is unprofitable for this model, but we expect that larger

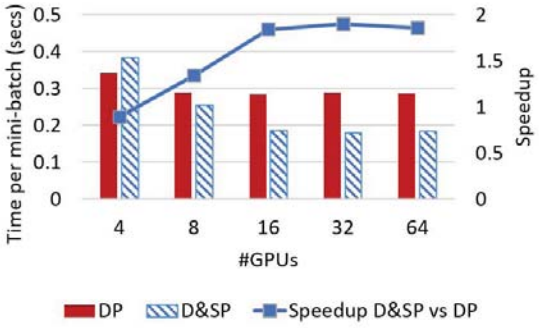


Fig. 16: Strong-scaling study with the original "Attention Is All You Need" Transformer. The global mini-batch size was 128 and all of the D&SP runs used topology-aware communication. We ran 2-, 4-, and 8-way D&SP and report the best result.

models would offer more room for scaling. Note that even with a moderate mini-batch size, D&SP continues to provide more efficient GPU utilization than DP. Figure 16 shows that with sixteen GPUs, there is a $1.52\times$ speedup versus DP.

G. Reducing Training Time

Finally, one of the critical challenges of accelerating the training of deep neural networks is to do so without degrading the learning of the model. Our algorithms for sub-graph parallelism result in an implementation that is mathematically equivalent to a non-parallel implementation, similar to how data parallelism does not impact a model's learning as long as the global mini-batch size is not changed. Figure 17 shows how the progression of the objective function is identical for 4-way D&SP and DP, demonstrating that the training is actually equivalent. It takes D&SP the same number of epochs to to achieve a fixed objective function as DP, and the wall-clock time is 35% shorter.

VII. Conclusion

Transformer-based neural network architectures are making breakthroughs in natural language processing and are showing great promise in scientific machine learning, but are extremely expensive to train. This paper presents a generalized approach to sub-graph parallelism that provides algorithmic optimizations over pure data parallelism. It allows for both strong- and

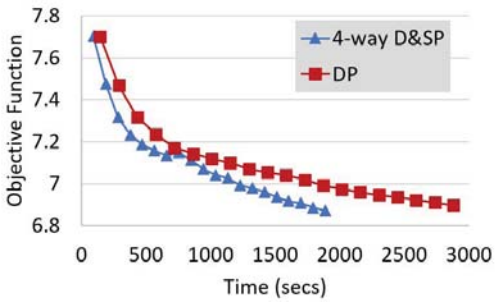


Fig. 17: Training the original "Attention is All You Need" Transformer on 64,000 samples from WMT14 with 16 GPUs. Batch size is 128 and the objective function is the average cross entropy loss on the reconstructed text.

weak-scaling when training Transformers, without requiring increases in model capacity to maintain compute efficiency. We provide a thorough discussion of the communication patterns and performance trade-offs, allowing these techniques to be applied to general multi-branch neural network architectures. We demonstrate how combining sub-graph parallelism with data parallelism and using a topology-aware layout allows a $1.84\times$ speedup when strong-scaling the original "Attention Is All You Need" Transformer and a $3.05\times$ speedup when weak-scaling a variant of T5-Large. Finally, we contribute these optimizations to an open-source distributed deep learning framework to enable the wider community to apply these algorithms for their own applications.

ACKNOWLEDGMENT

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-CONF-819861). This research was supported by the Exascale Computing Project (17-SC-20-SC). Effort from the Ohio State University has been supported in part by NSF grants #1931537, #2007991, #2018627, and XRAC grant #NCR-130002. Experiments were performed at the Livermore Computing facility.

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, 2019.

[3] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2020.

[4] 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," 2020.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[6] M. Chen, A. Radford, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *International Conference on Machine Learning*, 2020.

[7] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2019.

[9] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2020.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *arXiv preprint arXiv:1611.05431*, 2016.

[11] D. Amodei and D. Hernandez, "AI and compute," Sep 2020.

[12] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[13] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[14] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, *et al.*, "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in neural information processing systems*, pp. 103–112, 2019.

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

[16] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," 2019.

[17] N. Kitaev, Łukasz Kaiser, and A. Levskaya, "Reformer: The efficient transformer," 2020.

[18] Z. Jia, J. Thomas, T. Warszawski, M. Gao, M. Zaharia, and A. Aiken, "Optimizing DNN computation with relaxed graph substitutions," in *Proceedings of the 2nd Conference on Systems and Machine Learning (SysML)*, 2019.

[19] Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken, "TASO: optimizing deep learning computation with automatic generation of graph substitutions," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[20] R. A. Van De Geijn and J. Watts, "SUMMA: Scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[21] N. Dryden, N. Maruyama, T. Moon, T. Benson, M. Snir, and B. Van Essen, "Channel and filter parallelism for large-scale CNN training," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–20, 2019.

[22] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, "LBANN: Livermore big artificial neural network HPC toolkit," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[23] Lawrence Livermore National Laboratory, "Hydrogen: GPU-accelerated distributed linear algebra library." https://github.com/LLNL/hydrogen, 2020.

[24] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Trans. Math. Softw.*, vol. 39, Feb. 2013.

[25] Lawrence Livermore National Laboratory, "Lassen HPC System." https://hpc.llnl.gov/hardware/platforms/lassen, 2020.

[26] N. Dryden, N. Maruyama, T. Moon, T. Benson, A. Yoo, M. Snir, and B. Van Essen, "Aluminum: An asynchronous, GPU-aware communication library optimized for large-scale training of deep neural networks on HPC systems," in *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*, pp. 1–13, 2018.

[27] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. s. Tamchyna, "Findings of the 2014 workshop on statistical machine translation," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*, (Baltimore, Maryland, USA), pp. 12–58, Association for Computational Linguistics, June 2014.