Thorough Characterization and Analysis of Large Transformer Model Training At-Scale

SCOTT CHENG*, The Pennsylvania State University, USA
JUN-LIANG LIN, The Pennsylvania State University, USA
MURALI EMANI, Argonne National Laboratory, USA
SIDDHISANKET RASKAR, Argonne National Laboratory, USA
SAM FOREMAN, Argonne National Laboratory, USA
ZHEN XIE, Binghamton University, USA
VENKATRAM VISHWANATH, Argonne National Laboratory, USA
MAHMUT T. KANDEMIR, The Pennsylvania State University, USA

Large transformer models have recently achieved great success across various domains. With a growing number of model parameters, a large transformer model training today typically involves model sharding, data parallelism, and model parallelism. Thus, the throughput of large-scale model training depends heavily on the network bandwidth since a combination of model sharding and multiple parallelism strategies incurs various costs. However, prior characterizations of transformer models on high-bandwidth DGX machines that use TFLOPS as a metric may not reflect the performance of a system with lower bandwidth. Furthermore, data and model parallelism reveal significantly distinct training profiles on different system bandwidths at scale and, thus, need a thorough study.

In this paper, we provide a bottom-up breakdown of training throughput into compute and communication time, and quantitatively analyze their respective influences on overall end-to-end training scaling. Our evaluation involves an in-depth exploration of data parallelism, scaling up to 512 GPUs with limited bandwidth, and examines three model sharding strategies among six model sizes. We also evaluate three combinations of model parallelism on both high and low bandwidth supercomputing systems. Overall, our work provides a broader perspective on large-scale transformer model training, and our analysis and evaluation yield practical insights for predicting training scaling, shaping the future development of supercomputing system design.

CCS Concepts: • **Computing methodologies** → *Machine learning*; • **General and reference** → **Evaluation**; • **Networks** → Network performance evaluation.

Additional Key Words and Phrases: large language model

ACM Reference Format:
Scott Cheng, Jun-Liang Lin, Murali Emani, Siddhisanket Raskar, Sam Foreman, Zhen Xie, Venkatram Vishwanath, and Mahmut T. Kandemir. 2024. Thorough Characterization and Analysis of Large Transformer

*Work done during internship at Argonne National Laboratory.

Authors' addresses: Scott Cheng, The Pennsylvania State University, University Park, USA, contact@chengscott.io; Jun-Liang Lin, The Pennsylvania State University, University Park, USA, jpl6521@psu.edu; Murali Emani, Argonne National Laboratory, USA, memani@anl.gov; Siddhisanket Raskar, Argonne National Laboratory, USA, sraskar@anl.gov; Sam Foreman, Argonne National Laboratory, USA, foremans@anl.gov; Zhen Xie, Binghamton University, USA, zxie3@binghamton.edu; Venkatram Vishwanath, Argonne National Laboratory, USA, venkat@anl.gov; Mahmut T. Kandemir, The Pennsylvania State University, University Park, USA, mtk2@psu.edu.

ACM 2476-1249/2024/3-ART8
https://doi.org/10.1145/3639034

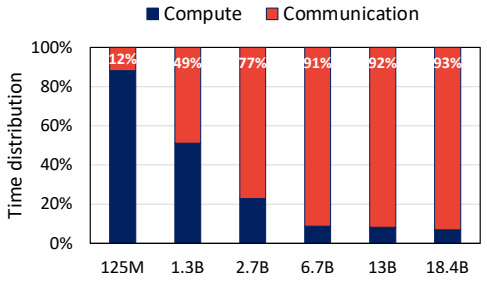


Fig. 2. Time distribution of compute and communication per training iteration for increasing transformer model sizes. [2]

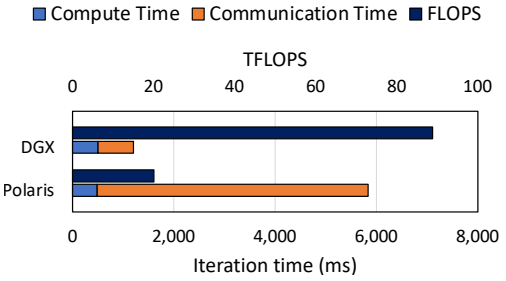


Fig. 3. FLOPS (floating-point operations per second) metric can not fully reflect the underlying execution profile, especially in a bandwidth-limited system as opposed to DGX systems. [3]

Model Training At-Scale. *Proc. ACM Meas. Anal. Comput. Syst.* 8, 1, Article 8 (March 2024), 25 pages. https://doi.org/10.1145/3639034

1 INTRODUCTION

With the emergence of ChatGPT [28] and the subsequent advancements of Copilots [13] and GPT4 [29], large transformer models have become the vanguard of the current AI revolution and catalyze the surge of generative AI exemplified by Stable Diffusion [38] and DALL-E [35]. Meanwhile, their capabilities are evolving beyond content generation, leading to multifaceted scientific breakthroughs, such as the protein structure prediction demonstrated in AlphaFold [19] and whole genome analyses during the COVID-19 pandemic showcased by GenSLM [52]. This widespread adoption of transformer models brings profound social and economic impacts. For example, a recent OpenAI study [12] reveals that about 80% of the U.S. workforce could undergo changes in at least 10% of their daily tasks due to the growing prevalence and integration of large language models (LLMs) in various sectors and industries. This insight emphasizes the increasing importance and transformative potential of introducing large transformer models. Consequently, given the emergent capabilities in transformer models [1, 45], the transformer model parameters are doubling every 4 months, as depicted in Fig. 1.

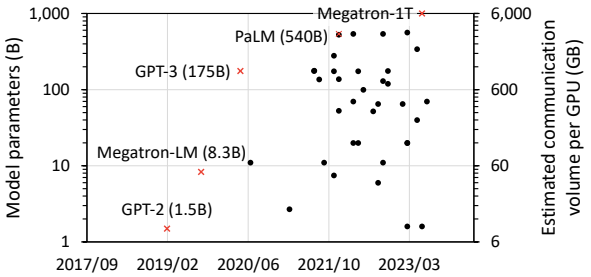


Fig. 1. The evolution of transformer model parameters and estimated per-GPU communication volume during training where names and cross marks present the state-of-the-art models at that time. [1]

[1]The figure is adapted from the data in [14, 16]. The communication volume is estimated to be three times the model parameter size, assuming ZeRO-3 data parallel training with a half-precision model weight. In practice, the actual communication volume typically is higher than our estimation due to the additional memory consumption in model parallelism.
[2]The figure exhibits evaluations of GPT models on Polaris with 16 GPUs.
[3]The figure presents evaluations of an 18.4B GPT model on 64 GPUs. The per-GPU FLOPS can be higher if the batch size is larger, but we use batch size one on both systems for illustration purposes.

Moreover, due to the growing number of parameters in state-of-the-art transformer models and the limited memory capacity per GPU, various parallelism and sharding strategies must be considered during the training process. For example, training a half-precision transformer model with the Adam optimizer [22] typically requires a memory capacity as much as 12 times the size of the model parameters. With appropriate data and model parallelism, we can both scale up the training of large transformer models and reduce memory usage per GPU. However, the substantial increase in model size amplifies the communication volume during end-to-end training. As shown in Fig. 2, the growth in transformer model size leads to a proportionally higher communication time during each training iteration. The need to communicate model weights and activations across the entire system during data and model parallelism training significantly stresses the system network bandwidth, and a system with a lower interconnection bandwidth may reshape the large transformer training landscape at scale. Additionally, different sharding strategies, including data and model parallelism, incur various communication volumes, leading to various training throughput. Data parallelism is the most commonly employed in training, and specifically, in the large transformer training context, data parallelism is typically combined with three kinds of model sharding levels. Overall, the increasing number of model parameters significantly impacts the overall system throughput, as the rise in communication volumes affects the end-to-end transformer model training throughput at-scale.

However, more than 40% of TOP500 supercomputers [26] are only equipped with an interconnect link speed of 100 Gb/s-equivalent or less[4], in contrast to an NVIDIA DGX machine that comes with eight Infiniband HDR (200 Gb/s) per node or higher in a newer generation. Therefore, recent studies [27, 34] on those high bandwidth servers does not fully capture the large transformer model training landscape in terms of training throughput and scalability. For instance, Fig. 3 shows that, only the TFLOPS metric, which is widely adopted and reported in many DGX system results, cannot fully reflect the overall training throughput. In contrast, a compute and communication breakdown provides a clear causality of the execution profile and thus gives reasoning for training the model at-scale. Moreover, since data-parallelism is the most common and applicable to every large-scale transformer model training, we mainly focus on characterizing the data-parallel large transformer model comprehensively, and our characterization complements prior works' study on sophisticated model parallelism and auto-parallelization strategies, as well as compute and memory optimizations. We study how the underlying system leads to compute and communication breakdown during model training, thereby contributing to the end-to-end training throughput.

Motivated by the observations above, in this paper, we characterize large transformer model training at scale using the state-of-the-art Megatron-DeepSpeed [3] training framework. Specifically, we compare how data parallelism scales across multiple nodes on three supercomputing systems, and provide a multi-node scalability estimation based on 8-GPU profiling and multi-node communication efficiency. In addition, we analyze three dimensions of model and data parallelism combinations under a limited network bandwidth. Thus, the main **contributions** of this work are as follows:

- We empirically evaluate end-to-end transformer training on both high-bandwidth (DGX) and low-bandwidth systems with various compute capabilities up to 512 GPUs, six model sizes up to 18.4B, 3 data parallel sharding strategies (ZeRO stages), 3 combinations of model parallelism, and 2 model architectures (GPT and BERT) that involve 4 kinds of collective communication calls, and provide three emulated limited network bandwidth cases in model parallel scaling.
- We provide a quantitative bottom-up analysis of compute and communication time estimation, as well as the scaling effects across varying numbers of nodes and sharding strategies. Our performance modeling is applicable to different model sizes and system architectures. The

[4]As of June 2023, mainly composed of 100GbE, SlingShot-11, Omni-Path, Infiniband EDR and their prior generations.

collected results indicate that our speedup prediction yields a mean squared error of less than 2.0%, compared to our evaluation results.
- Our evaluation results for data parallelism training indicate that a lower bandwidth system may result in a 7.35 times increase in communication time, consequently reducing the overall training throughput by 59.25%. This impact becomes much more significant in ZeRO-2, where lower network efficiency leads to even worse scaling on the lower bandwidth system with more GPUs. This observation was not emphasized in previous works since DGX machines with higher bandwidth exhibit near-linear speedups, as opposed to our lower bandwidth scaling.

The remainder of this paper is structured as follows. The following section introduces the transformer model, as well as data and model parallel training. Section 3 establishes the baselines for evaluation, and presents our scalability analysis. Following that, a bottom-up characterization from both compute and communication perspectives is conducted in Section 4, and specifically, a summary of our insights is given in Section 4.5. Finally, the related works are discussed in Section 5.

2 BACKGROUND

In this section, we introduce the transformer model architecture in Section 2.1. We then present large transformer model sharding strategies and their associated communication volume during training in Section 2.2. Further, Section 2.3 delves into data and model parallelism. Finally, Section 2.4 gives an overview of collective communication calls.

2.1 Transformer Model

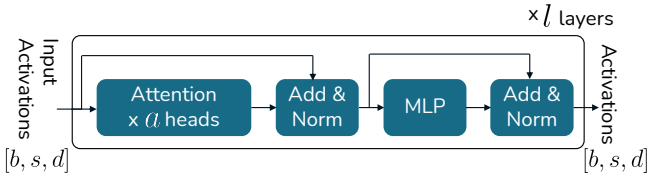


Fig. 4. Components of a transformer layer.

Model	Estimation
Parameters $\mathcal{P}$	$12d^2l$
FLOPs	$b(72sd^2 + 12s^2d)l$
Activation (bytes)	$b(34sd + 5as^2)l$

Table 1. Estimation of parameters, FLOPs, and activation memory of a transformer model.

A transformer model [42] consists of multiple transformer layers and each transformer layer, depicted in Fig. 4, comprises multi-head attention, MLP (multi-layer perception, pointwise feed-forward network), normalization, and residual connections between them. To describe a transformer model, we use the following six notations in the rest of this paper:

- model parameters: $\mathcal{P}$
- the number of transformer layers: l
- the number of attention heads: a
- batch size: b
- input sequence length: s
- hidden dimension: d

The activations refer to intermediate tensors during forward propagation that will be used for gradient calculations during the backward pass. For instance, the output tensor from multi-head attention, a temporary value during forward pass, is also counted as an activation since it will be used to compute gradient during backward propagation. Additionally, we refer to the size of activation as activation memory in the rest of the paper. For an input batch size b, the input activation of each transformer layer is a tensor of dimension $[b, s, d]$. Additionally, Table 1 shows that we can estimate the model parameters, FLOPs (floating-point operations), and activation size accordingly [23].

Further, GPT [1] and BERT [21] are two representative transformer model architectures built upon transformer layers. Both models first encode the input sequence by word embedding and positional encoding. Then, the encoded sequence propagates through multiple transformer layers. The difference between the two models is that the GPT employs masked attention that restricts visibility only to partial input, as opposed to BERT, which can capture the entire sequence in each layer. The output of the transformer layer can be used for probability prediction or completing the missing tokens in the input sequence.

2.2 Transformer Model Sharding: ZeRO

During transformer model training, a $\mathcal{P}$-parameter model needs to consume $16\mathcal{P}$ bytes of memory when using the Adam optimizer [22]. Thus, the ZeRO (Zero Redundancy Optimizer) [33] technique has been proposed to address the challenge of accommodating an increasing number of model parameters into limited GPU memory by sharding model components across multiple GPUs. ZeRO offers three stages, namely, ZeRO-1, ZeRO-2, and ZeRO-3, each involving a different level of model sharding. As the sharding level increases, more parameters are sharded, which leads to higher communication costs but reduces per GPU memory consumption. Table 2 lists the model components sharded across multiple GPUs and the associated per GPU communication volume of each ZeRO stage.

	Sharded components	Communication volume (bytes)
ZeRO-1	optimizer state	$4\mathcal{P}$
ZeRO-2	+ gradient	$4\mathcal{P}$
ZeRO-3	+ model weight	$6\mathcal{P}$

Table 2. Sharded model components and communication volume per GPU for each ZeRO stage.

Fig. 5 illustrates the training process during each iteration for the three ZeRO stages. In ZeRO-1, each process performs forward and backward passes in parallel. Once the local gradients are computed and available, all processes engage in the AllReduce operation to calculate global gradients. Subsequently, the optimizer updates the weights locally using the global gradients. Each process then participates in the AllGather operation to collect the updated model weights from all GPUs. Since both the gradients and the weights consume each $2\mathcal{P}$ bytes, the total communication volume amounts to $4\mathcal{P}$ bytes, as indicated in Table 2. Moreover, in practice, the backward and optimize phases are often combined as a pipeline since a process can start collective communication calls once partial gradients are available. Thus, the figure only depicts the equivalent sequential functionality.

Similarly, the overall training process in ZeRO-2 is similar to ZeRO-1, with one key difference: the gradients are sharded, and thus, the global gradient only needs to be computed by ReduceScatter. In practice, the ReduceScatter operation may be implemented using multiple Reduce operations to effectively hide latency as shown in the figure. On the other hand, in ZeRO-3, besides optimizer states and model gradients, the model weights are also sharded across multiple GPUs. Consequently, during both forward and backward propagation, the model weights must be gathered via AllGather, incurring an additional communication cost of $4\mathcal{P}$ bytes. Then, the global gradient is computed by ReduceScatter, similar to ZeRO-2. Hence, the communication volume per GPU is $6\mathcal{P}$ bytes, and the total communication volume across the g-GPU system becomes $6g\mathcal{P}$ bytes.

2.3 Data and Model Parallelism in Transformer Model

Data parallelism (DP) and model parallelism (MP) are typically employed in the transformer model training. Moreover, Megatron-DeepSpeed [3] is the state-of-the-art framework offering both data and model parallelism implementations integrated with ZeRO techniques for large transformer model training. Unlike data parallelism discussed in the prior section, which partitions the input

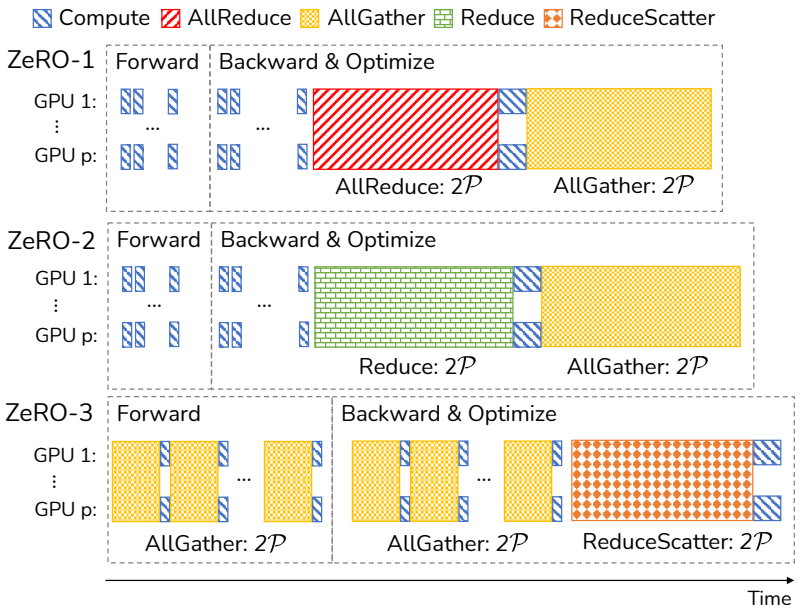


Fig. 5. Compute and collective communication calls with their associated communication volume (in bytes) per training iteration for each ZeRO stage, assuming a transformer model of parameter size $\mathcal{P}$. Specifically, collective communication calls include (1) AllReduce operation sums the local data from each GPU and distributes the summed result back to all GPUs. (2) AllGather operation combines local data from all GPUs to each GPU. (3) Reduce operation sums the local data from all GPUs but only places the summed result in a specific GPU. (4) ReduceScatter operation is equivalent to performing Reduce but scattering different portions of the reduced data to each GPU.

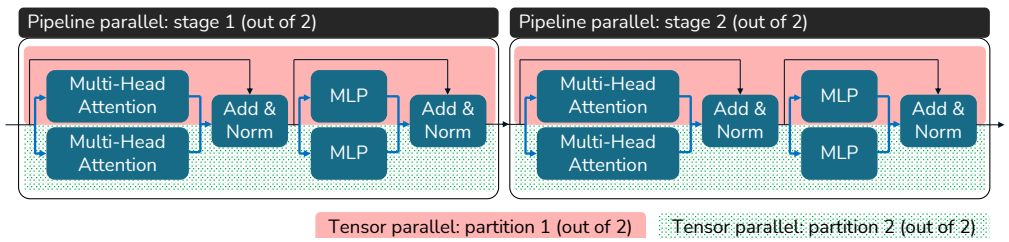


Fig. 6. (pipeline parallelism, tensor parallelism) = (2, 2).

data, model parallelism instead partitions the model activations and comprises pipeline parallelism (PP) and tensor parallelism (TP) [39].

Fig. 6 illustrates two transformer layers with two pipeline parallelism stages and two tensor parallelism partitions, with the blue arrows indicating that the activations are sharded or combined across GPUs. Typically, tensor parallelism is performed within a node (intra-node), whereas pipeline parallelism is distributed across nodes (inter-node). With a training scheme that considers these three types of parallelism, the global batch size (GBS) can be computed as follows:

$$\#\text{DP} \times \#\text{TP} \times \#\text{PP} \times \text{MBS} = \text{GBS}, \tag{1}$$

where MBS denotes micro batch size on each GPU.

2.4 Collective Communication

Different collective communications on the same size of data may incur a different amount of data transferred in the network. Table 3 shows the estimated per GPU communication volume incurred in the network when performing a collective communication of a model size of $2\mathcal{P}$ bytes on p GPUs [4, 47]. For instance, AllReduce needs to perform $2(p-1)$ data transfers among p GPUs, resulting in a communication volume of $\frac{2(p-1)}{p}2\mathcal{P} = 4(1-\frac{1}{p})\mathcal{P}$ at best. However, because various collective algorithms exist, such as ring-based and tree-based implementations, the latencies may vary, and the actual communication volumes may be higher than the estimations. Furthermore, since we focus on large-scale systems, which typically involve more than 32 GPUs ($p \geq 32$), the term $(1-1/p)$ only affects the estimation by less than 3.1%. Hence, it can be ignored in most of the following discussions.

Operation	Communication volume (bytes)
AllReduce	$4(1-\frac{1}{p})\mathcal{P}$
AllGather	$2(1-\frac{1}{p})\mathcal{P}$
ReduceScatter	$2(1-\frac{1}{p})\mathcal{P}$
Reduce	$2(1-\frac{1}{p})\mathcal{P}$

Table 3. Estimated per GPU communication volume for various collective communication calls for a model size of $2\mathcal{P}$ bytes on p GPUs.

Regarding the comparisons between collective operations, AllReduce requires double the volume compared to AllGather and ReduceScatter. This is because the AllReduce operation can be decomposed into an AllGather and a ReduceScatter operation. Although, in practice, AllReduce is faster than ReduceScatter+AllGather due to further optimizations in the implementation, the latter scheme is still used because it can better integrate with the training pipeline of transformer models and improve the overall training throughput.

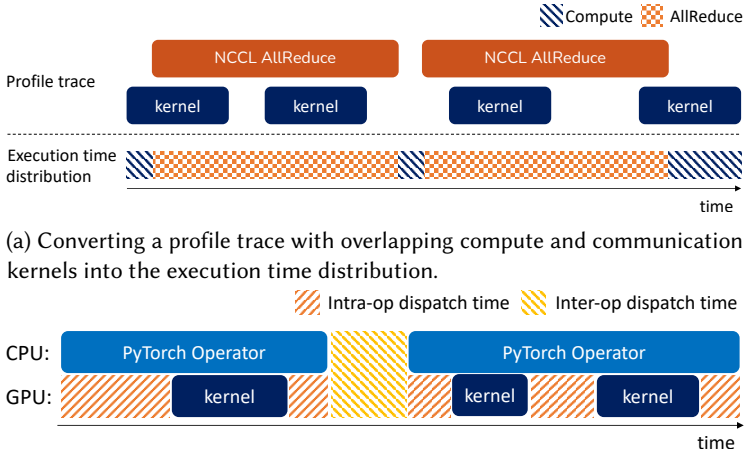
3 METHODOLOGY

In this section, we first outline our approach to profiling via instrumentation, detailed in Section 3.1. Next, we introduce the multi-node scalability analysis based on our profiling results in Section 3.2. Finally, we establish the evaluation baselines in Section 3.3.

3.1 Instrumentation

To gain a deeper insight into the compute and communication patterns exhibited by large-scale transformer model training, we integrate additional NVTX [8] instrumentation provided by PyTorch [30] into the Megatron-Deepspeed [3] framework. The instrumentation provides a granular view of compute and communication behavior of kernels for each training stage, namely, the forward, backward, and optimization stages. We first train the model for several iterations and then retrieve the runtime information from the profiling database generated by NVIDIA Nsight [6], including CUDA kernels, events, and OS runtime. Based on the NVTX instrumentation information, we can associate each CUDA kernel or function call to its corresponding training phases.

Further, since a compute kernel might run concurrently with another communication kernel, we illustrate this scenario with a sample profiling trace in Fig. 7a. The top of the figure presents the trace for a single GPU consisting of two NCCL AllReduce operations along with four compute kernels. Given that the compute and communication kernels might overlap with each other, we adopt the convention in PyTorch profiler [25], to first represent the communication kernel in the execution time distribution, followed by the compute kernel, as shown at the bottom of Fig. 7a. As a result, the actual compute duration might be longer than the time shown in the distribution. Nonetheless, it also indicates no communication kernels will occur during the compute time. Specifically, the compute time includes both GPUs and CPUs, such as PyTorch operators' dispatch time. In this



(a) Converting a profile trace with overlapping compute and communication kernels into the execution time distribution.

(b) Dissect compute time into kernel execution, intra-op, and inter-op dispatch time.

Fig. 7. Top-down interpretation of the profile traces collected by our instrumentation.

example, the time between two NCCL AllReduce operations is marked as “compute" since the gap typically means the CPU is launching the GPU kernel.

In addition to communication kernels, we also factor in the “overhead" associated with the GPU compute that contributes to the overall training throughput. This is especially pertinent when dealing with smaller model sizes, where these overheads are non-negligible. Quantitatively, we classify the compute time into intra-operator (intra-op) and inter-operator (inter-op) dispatch time, and kernel execution time. For instance, Fig. 7b illustrates a typical execution timeline consisting of two PyTorch operators. During the transformer model training, the main program in the CPU initiates a sequence of PyTorch operators, such as matrix multiplications and multi-head attentions, and each operator may subsequently launch zero to a few GPU kernels. Effectively, the actual compute time is the sum of all GPU kernels, but there might be overhead interleaved among the GPU kernel and PyTorch operator launches. Thus, we define intra-op dispatch time as the time within each PyTorch operator, excluding the GPU kernel execution time; similarly, we define inter-op dispatch time as the time between two consecutive PyTorch operators. A more in-depth analysis of compute dispatch time is given in Section 4.1.

3.2 Scalability Analysis

In general, data parallelism is more scalable than model parallelism, especially in the scale of hundreds to thousands of GPUs. This is due to the fact that, in data parallelism, only model weights and gradients are communicated, whereas model parallelism transmits the activation memory. As detailed earlier in Section 2.1, the activation memory usually surpasses the size of model weights, and, more importantly, the activation memory increases in proportion to the input batch size. Hence, our subsequent analysis will focus on examining the scalability of data parallelism for a given model size. To project the multi-node training throughput, we incorporate the profiling results from both the single-node training and multi-node network efficiency. As discussed in the previous section, the duration of each training iteration is the sum of the compute and communication times. For instance, the following formulation shows the per batch training speedup when increasing the

GPU count from 8 to 16:

$$\text{speedup}_{8\to16} = \frac{t^{(8)}_{\text{iter.}}}{t^{(16)}_{\text{iter.}}} = \frac{t^{(8)}_{\text{compute}} + t^{(8)}_{\text{comm.}}}{t^{(16)}_{\text{compute}} + t^{(16)}_{\text{comm.}}}. \tag{2}$$

where $t^{(8)}_{\text{iter.}}$ represents the per-iteration training latency on 8 GPUs, and $t^{(8)}_{\text{compute}}$ and $t^{(8)}_{\text{comm.}}$ denote, respectively, the per-GPU compute and communication times with 8 GPUs. Similar notation applies to the 16-GPU setup as well. In general, we introduce the multi-node network efficiency when scaling from baseline g GPUs to n GPUs, as:

$$\text{eff}_{g\to n} = \frac{t^{(g)}_{\text{comm.}}}{t^{(n)}_{\text{comm.}}}. \tag{3}$$

Thus, the following formula estimates the "end-to-end" speedup from baseline g GPUs to n GPUs:

$$\text{speedup}_{g\to n} = \frac{t^{(g)}_{\text{iter.}}}{t^{(n)}_{\text{iter.}}} = \frac{t^{(g)}_{\text{compute}} + t^{(g)}_{\text{comm.}}}{t^{(n)}_{\text{compute}} + t^{(n)}_{\text{comm.}}} = \frac{r^{(g)}_{\text{compute}} + r^{(g)}_{\text{comm.}}}{r^{(g)}_{\text{compute}} + r^{(g)}_{\text{comm.}}/\text{eff}_{g\to n}}. \tag{4}$$

where $r^{(n)}_{\text{compute}} = t^{(n)}_{\text{compute}}/t^{(n)}_{\text{iter}}$ and $r^{(n)}_{\text{comm.}} = t^{(n)}_{\text{comm.}}/t^{(n)}_{\text{iter}}$. It is to be noticed that obtaining the compute and communication time distribution per iteration on the baseline system, namely, $r^{(g)}_{\text{compute}}$ and $r^{(g)}_{\text{comm.}}$, and the network efficiency is sufficient to estimate the overall speedup. For instance, from the 8-GPU profiling with instrumentation, we can determine $(r^{(8)}_{\text{compute}}, r^{(8)}_{\text{comm.}})$, and, based on multi-node efficiency analysis, we can obtain $\text{eff}_{8\to16}$. A detailed efficiency evaluation is given in Section 4.2.

3.3 Experimental Methodology

In this section, we introduce the hardware, software, model, and evaluation setups to establish the evaluation baselines.

Hardware Platform: We evaluate three GPU supercomputing systems, each with varied compute and networking setups, as detailed in Table 4. Specifically, TG40 and TG80 are NVIDIA DGX-based high bandwidth systems, and these three systems provide 160 GB, 320 GB, and 640 GB GPU memory per node accordingly. The number of nodes used for scaling evaluation (#Nodes scaled) is chosen by considering scheduling availability, constraints, and out-of-order nodes. Moreover, compared to Polaris, both TG40 and TG80 have double the number of GPUs per node and exhibit a higher overall system bandwidth due to a newer generation of compute NICs and an increased number of NICs per node. To elaborate, NVIDIA Mellanox ConnectX-5 and ConnectX-6 deliver networking bandwidths of 100 Gb/s and 200 Gb/s, respectively.

To figure out the "achievable" bandwidth in the GPU systems, we first perform bandwidth evaluations using `bandwidthTest` and `p2pBandwidthLatencyTest` from `cuda-samples` [9] on all GPU systems. Table 4 shows the average memory copy bandwidth for transferring 1GB of data[6] between the host and GPU memory and the average bidirectional peer-to-peer (P2P) bandwidth for communication between two distinct GPUs within a node. Specifically, the host memory used in memory copy could either be pinned or pageable. The table reveals that while the pinned memory copy reflects that the achievable PCIe bandwidth is around 25 GB/s, as opposed to the 32 GB/s link speed, the pageable memory copy represents the bandwidth for the most common GPU memory operations. When comparing the P2P bandwidth, TG40 and TG80 have three times

[5]ThetaGPU consists of two kinds of system configurations; thus, we distinguish by the two abbreviations: TG40 and TG80.
[6]The choice of a 1GB data size is based on the largest common memory operations during training and thus may not be able to fully utilize the entire link bandwidth due to the chosen data size.

	Polaris	TG40[5]	TG80[5]
System	HPE Apollo	NVIDIA DGX	NVIDIA DGX
Nodes	560	22	2
#Nodes (#GPU) scaled	128 (512)	8 (64)	2 (16)
CPU Model	AMD 7543P	AMD 7742	AMD 7742
CPU Socket(s)	1	2	2
GPU	NVIDIA A100	NVIDIA A100	NVIDIA A100
per GPU Memory	40GB HBM2	40GB HBM2	80GB HBM2e
#GPU per node	4	8	8
GPU Memory B/W	1555 GB/s	1555 GB/s	2039 GB/s
#NVLink per GPU	12 (4 per peer)	12 (NVSwitch)	12 (NVSwitch)
Compute NIC	ConnectX-5	ConnectX-6	ConnectX-6
#Interconnect per node	2	8	8
Total NIC B/W per node	200 Gbps (25 GB/s)	1.6 Tbps (200 GB/s)	1.6 Tbps (200 GB/s)
pinned memory copy B/W	24.6 GB/s	26.1 GB/s	26.2 GB/s
pageable memory copy B/W	19.2 GB/s	12.2 GB/s	12.4 GB/s
P2P B/W	80.5 GB/s	277.6 GB/s	278.8 GB/s

Table 4. The GPU supercomputing systems evaluated in this study.

higher bandwidth than Polaris since both TG40 and TG80 have 12 NVLinks (300 GB/s link speed) per GPU, whereas Polaris only has 4 NVLinks (100 GB/s link speed) per peer. These benchmark results serve as the foundational insights for our subsequent evaluations.

Software Setup: We evaluate transformer model training on a state-of-the-art framework – Megatron-DeepSpeed [3]. For all our evaluations, we use the fp16 precision training, PyTorch implementation for data-parallelism, FlashAttention [11], and uniform activation recomputation. In addition, each process handles one GPU with micro-batch size one, meaning that the global batch size is the same as the total GPU count. On the other hand, Megatron-DeepSpeed incorporates NCCL 2.18 [5] for collective communication across the multi-node and multi-GPU environments. In addition to the process affinity mentioned earlier, we configure the `NCCL_NET_GDR_LEVEL` environment variable to `SYS` to utilize a higher bandwidth in the system topology [5].

To have a better understanding of the communication performance, we evaluated AllReduce with a half-precision data type, which is the most common collective communication primitive and data type during training, on `nccl-tests` [4] shown in Fig. 8. Within one node, the AllReduce bandwidth is around 200 GB/s on all three systems, which is bounded by the link speed of NVLink (300 GB/s), or specifically, the P2P bandwidth (277.6 GB/s) shown in Table 4. But, as the evaluation scales to 2 nodes, only TG40 and TG80 remain at high bandwidth, while, in contrast, the Polaris AllReduce bandwidth drops to around 20 GB/s. Similarly, the 4-node Polaris bandwidth is also around 20 GB/s, indicating that the bandwidth is bounded by

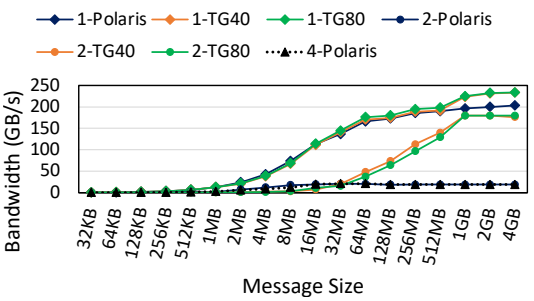


Fig. 8. The NCCL AllReduce bandwidth benchmark with the half-precision data type, where the number in each legend means the number of nodes for that GPU system.

Model	125M	1.3B	2.7B	6.7B	13B	18.4B
Layers (l)	12	24	32	32	40	40
Hidden dim. (d)	768	2064	2560	4096	5120	6144
Attention heads (a)	12	24	32	32	40	48
Sequence length (s)	1024	1024	1024	1024	1024	1024
Est. FLOPs (TFLOP)	0.6	8.2	16.5	41.2	79.9	114.4
Est. activation memory (GB)	2.4	12.1	21.7	33.3	57.0	74.0

Table 5. Transformer model configurations.

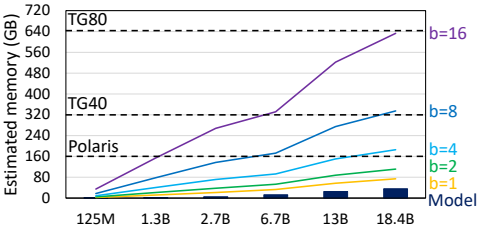


Fig. 9. Estimated memory usage for the model weights and activation memory under different batch sizes (b).

the compute NICs link speed (25 GB/s). In contrast, TG40 and TG80 can still achieve around 180 GB/s interconnection bandwidth.

Model Setup: For the following evaluations, we will mainly focus on the GPT model, unless otherwise specified. Due to the differences between causal trainings in BERT and GPT models, while FlashAttention [11] claims that it can theoretically achieve a 2x speedup with causal training, the end-to-end training time cannot attain this speedup since the FlashAttention kernel only contributes to the computation. Furthermore, we did not observe significant differences in our profiling results. We discuss the detailed similarities and differences in Sec 4.3. Table 5 lists six model configurations, with model sizes ranging from 125M to 18.4B, which are chosen based on prior works [1] with some adaption to align with the capabilities of Megatron-DeepSpeed framework. Further, the estimated floating-point operations (FLOP) and activation memory are estimated according to Section 2.1, assuming a micro-batch size of one. Fig. 9 illustrates the projected memory consumption based on model size and activation memory with varying batch sizes. In general, we will use micro-batch size one, and global batch size equals to the number of GPUs.

Evaluation Setup: The following execution times represent the average values over 100 training iterations, and the standard deviation presents the evaluation variability. We denote OOM as out-of-memory during training. The scalability plots are normalized against the first available result in that group of experiments. For example, if a model encounters OOM on 8 GPUs but is able to train on 16 GPUs, the scalability in that group of experiments is normalized against the 16 GPU result.

4 CHARACTERIZATION OF TRANSFORMER MODEL TRAINING

In this section, we evaluate and analyze compute in Section 4.1 and multi-GPU communication efficiency in Section 4.2. Section 4.3 thoroughly evaluates data-parallel training and combines to our prior analysis. Similarly, Section 4.4 evaluates model parallel training under three bandwidth-limited cases. Finally, based on our characterization results, Section 4.5 summarizes our insights on large transformer model training.

4.1 Computation

As discussed earlier in Section 3.1, we break down the compute time to further quantify the overhead or, more precisely, the operator dispatch time. Fig. 10 shows the compute time distribution over different model sizes on Polaris, where each compute time component corresponds to Fig. 7b. Notably, for a smaller model size, the dispatch time dominates about 95% of the compute time.

As the model size increases, the proportion of compute kernel time also grows, and the intra-op dispatch time spans from 78% down to 14%. Table 6 presents the time distribution of the most time-consuming operators, which include GPU compute kernels and intra-op during the 18.4B model training with ZeRO-3 enabled. This table indicates that the primary compute operators

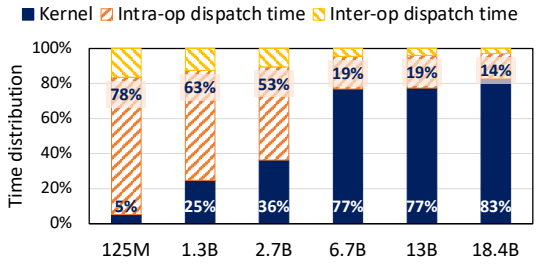


Fig. 10. Dissection of compute time per iteration.

	Operator	Time (%)
kernel	GEMM	59.5%
	CatArrayBatchedCopy	26.5%
	Elementwise kernel	9.6%
	Multihead attention	2.7%
intra-op	_post_forward_module_hook	28.1%
	PreBackwardFunction	26.8%
	PostBackwardFunction	24.3%
	reshape	6.4%
	permute	4.1%

Table 6. Time distribution for individual compute kernel and intra-op.

are GEMM (General Matrix Multiplication) and tensor concatenation, as the transformer model employs linear layers and merges tensors from multiple GPUs. It also highlights that the intra-op dispatch time involves model parameter sharding and data layout transformation, such as reshaping, permuting, and transposing, while the inter-op dispatch time typically consists of pthread switching or network polling. Based on the compute time profiling and the transformer model FLOPs listed in Table 1, we can estimate the compute time for the transformer model by:

$$t_{\text{compute-transformer}} = \frac{\text{Model FLOPs}}{\text{FLOPS}} = \frac{b(72sd^2 + 12s^2d)l}{\text{FLOPS}}, \tag{5}$$

where FLOPS represents the GPU floating point operations per second. Overall, to account for compute dispatch time, we factor in the kernel time distribution within compute time (k_1) and the transformer kernel time distribution within the kernel time (k_2):

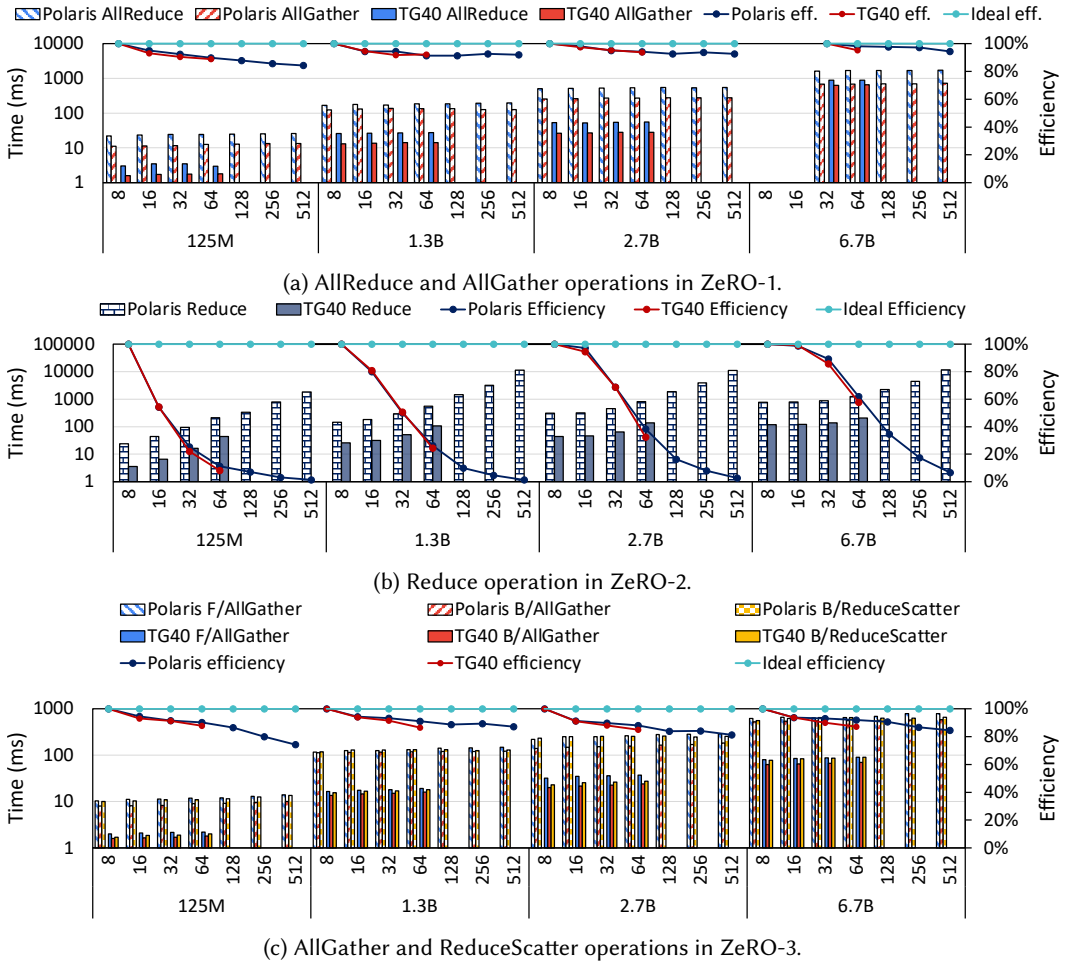
$$t_{\text{compute}} = \frac{t_{\text{compute-transformer}}}{k_1 \times k_2}. \tag{6}$$

For instance, for the 18.4B model, $k_1 \approx 83\%$ from Fig. 10 and $k_2 \approx 1 - 26.5\%$ from Table 6. Although dispatch time does consume a significant portion of the time, especially in smaller models, certain dispatch times, such as synchronization, are inevitable. Therefore, we refer to it as "dispatch time" rather than "overhead".

4.2 Communication

In this section, we delve into how the achievable hardware communication bandwidth, previously discussed in Section 3.3, factors into the collective communications scaling from 8 to 512 GPUs. When the model size grows, the corresponding communication volume increases, and thus, it becomes crucial to understand the scalability of different communication operations across multiple nodes. As discussed earlier, a model with $\mathcal{P}$ parameters trained using the ZeRO-1 sharding strategy will result in $4\mathcal{P}$ bytes of communication volume per GPU due to the AllReduce operation on the gradients and the AllGather operation on the model weights. Similarly, when using ZeRO-2 and ZeRO-3, the communication volume per GPU depends only on the model size, and is independent of the number of nodes.

Fig. 11 shows the average efficiency of the representative collective communication calls for each ZeRO stage on Polaris and TG40 across different number of GPUs, with the ideal efficiency being 100%. Due to the OOM issue, the figure only includes four model sizes, but the omitted results of larger models still exhibit similar scaling behavior. In addition, the scaling experiments on the TG40 system are limited to 64 GPUs, as discussed in Section 3.3.



(a) AllReduce and AllGather operations in ZeRO-1.

(b) Reduce operation in ZeRO-2.

(c) AllGather and ReduceScatter operations in ZeRO-3.

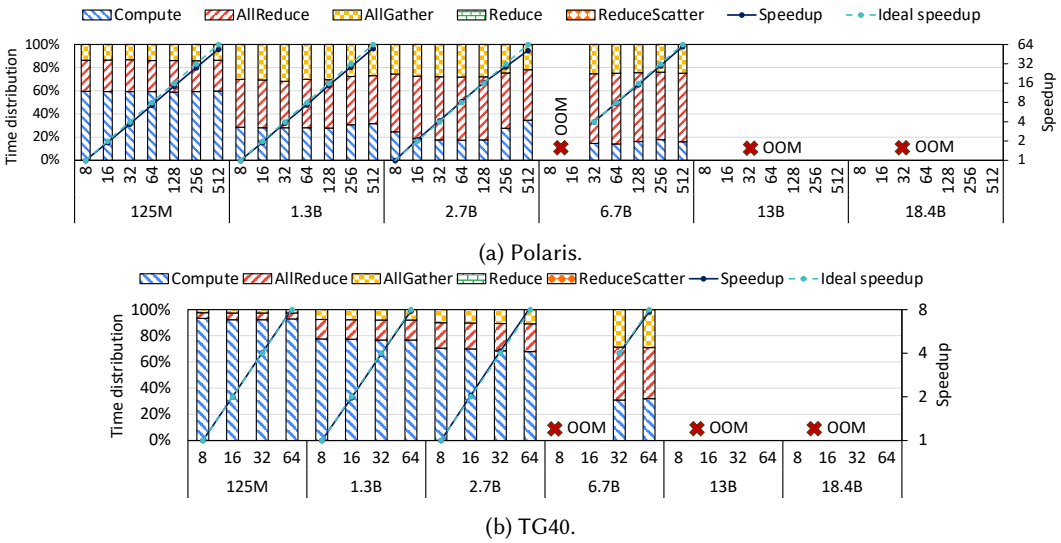
Fig. 11. Efficiency and log scale per GPU execution time of communication for a different number of GPUs, model parameters, and each ZeRO stage on Polaris and TG40.

Fig. 11a presents the efficiency and execution time of AllReduce and AllGather operations in ZeRO-1. The evaluation results align with the estimation derived from the prior discussion in Sections 2.1 and 2.3. Specifically, the AllReduce time can be estimated as follows:

$$t^{(p)}_{\text{AllReduce}} = 4\mathcal{P} \;/\; (BW^{(8)}\text{eff}_{8\to p}) \approx 4 \times 12d^2l \;/\; (BW^{(8)}\text{eff}_{8\to p}), \tag{7}$$

where $t^{(p)}_{\text{AllReduce}}$ represents the AllReduce communication time for a $\mathcal{P}$-parameter fp16 model, $BW^{(8)}$ denotes the achievable bandwidth on a system with 8-GPUs as discussed in Section 3.3, and $\text{eff}_{8\to p}$ denotes the multi-GPU scaling efficiency from 8 to p GPUs discussed in the prior section.

Similarly, we can deduce the estimated communication time for AllGather, which is half of the AllReduce time. The results shown in the figure regarding the AllGather and AllReduce communication times align with our estimations. Additionally, for the same model size, the communication time on TG40 is an order of magnitude lower than on Polaris since the network bandwidth is 8 times higher than that on Polaris, as shown in Table 4. It is to be noted however that, when



(a) Polaris.

(b) TG40.

Fig. 12. End-to-end per-iteration training time distribution and speedup for different number of GPUs, model parameters using ZeRO-1 stage.

comparing the communication efficiency of both Polaris and TG40, they exhibit a similar trend, and intuitively, as the number of GPUs increases, the efficiency drops to around 89.4% on average.

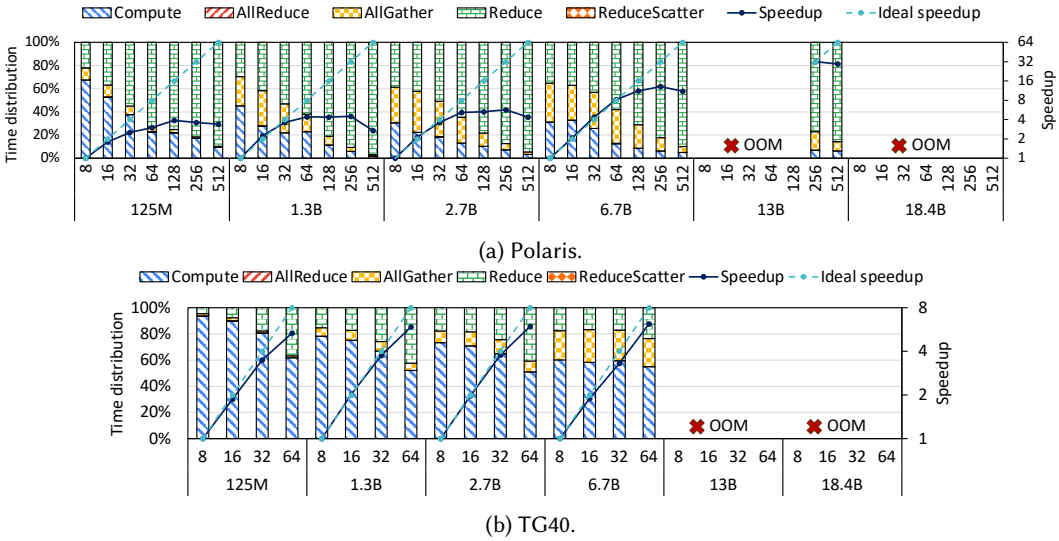
Fig. 11b shows the average Reduce time per ZeRO-2 training iteration. The AllGather operation is omitted since it has the same behavior as in ZeRO-1. In ZeRO-2, ReduceScatter is performed during the backward passes instead of AllReduce, and ReduceScatter is decomposed into multiple Reduce operations as discussed previously in Section 2.2. Similar to ZeRO-1, the communication volume of the Reduce operation is $2\mathcal{P}$ bytes. However, in contrast to AllReduce, the Reduce operation involves multiple GPU peer-to-peer operations [47]. Consequently, the communication time for Reduce increases as the number of nodes increases, resulting in a drastic efficiency drop.

Moreover, though the communication time on TG40 is significantly lower than on Polaris, both machines share a similar scaling trend in efficiency. Fig. 11c indicates that as the number of nodes increases, the communication efficiency of AllGather drops to around 80% in ZeRO-3. Furthermore, since the communication volumes of AllGather in both forward and backward passes and ReduceScatter in backward passes are the same, both being $2\mathcal{P}$ bytes, the communication time is also similar.

It is worth noting that several factors can influence the observed throughput and efficiency. For instance, our submitted job might be scheduled to scatter across multiple racks, and it might collocate with other jobs in the same rack. Therefore, considering that the network infrastructure is a shared resource among supercomputing systems, multiple Infiniband switching QoS may generate overheads and lead to an efficiency drop. Furthermore, non-deterministic decisions in communication, specifically the routing path between two GPUs, might lead to fluctuations in our measurements. Overall, the inefficiency in communication will impede the end-to-end training, and we will discuss this further in the next section.

4.3 Data Parallelism

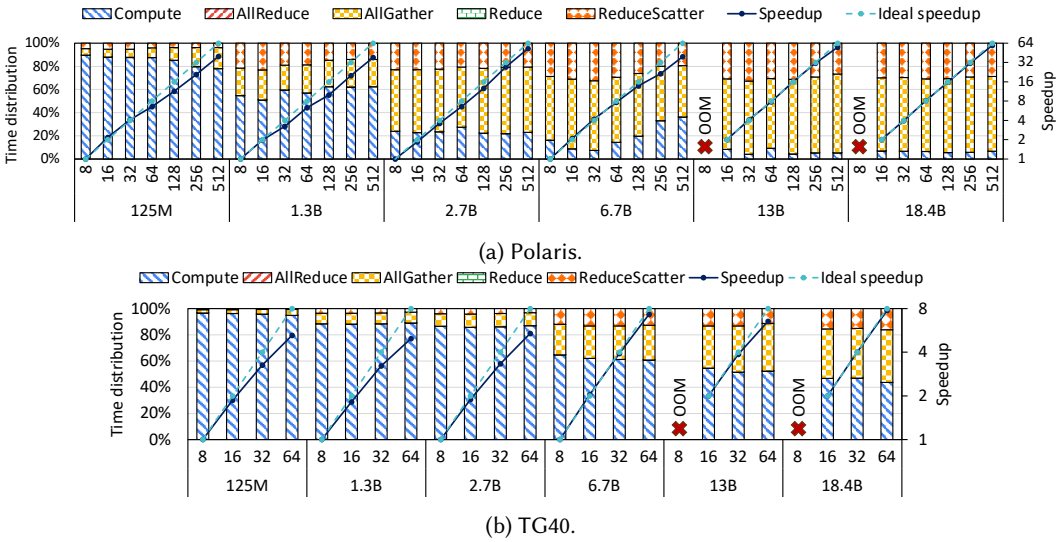
In this section, we discuss how communication efficiency contributes to overall training throughput during data parallelism training. Fig. 12 illustrates the scaling trends of the ZeRO-1 stage with



(a) Polaris.

(b) TG40.

Fig. 13. End-to-end per-iteration training time distribution and speedup for different number of GPUs, model parameters using ZeRO-2 stage.



(a) Polaris.

(b) TG40.

Fig. 14. End-to-end per-iteration training time distribution and speedup for different number of GPUs, model parameters using ZeRO-3 stage.

different numbers of GPUs and model parameters on Polaris and TG40 systems. As shown in Fig. 12a, the distribution of compute and communication remains at a fixed ratio under the same model size as the number of GPUs increases, since the per-GPU computations and communication volume remain constant as we keep the micro-batch size equal to 1. Based on the communication efficiency shown in Fig. 11a, the AllReduce and AllGather communications exhibit near-linear scalability, with efficiency dropping slightly to 89% when scaling to 512 GPUs. As a result, taking

	125M					1.3B					2.7B				
GPT	C	AR	AG	Total±Std	TFLOPS	C	AR	AG	Total±Std	TFLOPS	C	AR	AG	Total±Std	TFLOPS
Polaris	49.0	22.0	11.1	82.1±5.6	11.0	118.0	168.9	124.2	411.1±8.0	15.5	246.7	503.0	256.0	1005.7±7.5	16.4
TG40	63.6	3.0	1.6	68.2±3.8	13.3	135.3	26.0	13.0	174.3±9.5	44.3	191.3	53.0	26.5	270.8±7.2	57.8
TG80	63.1	2.4	1.2	66.7±3.6	13.6	135.3	23.3	12.6	171.2±6.5	45.8	191.3	40.3	21.9	253.5±5.9	60.3
BERT	C	AR	AG	Total±Std	TFLOPS	C	AR	AG	Total±Std	TFLOPS	C	AR	AG	Total±Std	TFLOPS
Polaris	56.3	21.1	8.9	86.4±4.4	10.1	118.0	165.5	93.5	377.0±8.8	14.1	246.7	493.1	248.1	988.0±7.8	18.9
TG40	63.6	2.4	1.2	67.2±2.9	13.0	135.3	26.0	13.0	174.3±8.7	37.9	211.3	39.9	21.6	272.8±5.9	61.3
TG80	63.1	2.4	1.2	66.7±3.9	12.7	135.3	22.9	12.4	170.6±5.5	39.2	199.5	39.8	21.5	260.9±6.3	64.4

Table 7. Breakdown of GPT and BERT model training times (ms) per iteration in the ZeRO-1 stage, along with the corresponding achieved FLOPS per GPU, across various model parameters in Polaris, TG40, and TG80. The abbreviations are as follows: C: Compute, AR: AllReduce, AG: AllGather.

into account the communication time distribution during the training iteration, ZeRO-1 can achieve a near-ideal speedup in the evaluation. A similar observation can be made for TG40, as shown in Fig. 12b, where ZeRO-1 also exhibits almost perfect linear scaling – in both computation and communication – on TG40, resulting in an overall optimal speedup.

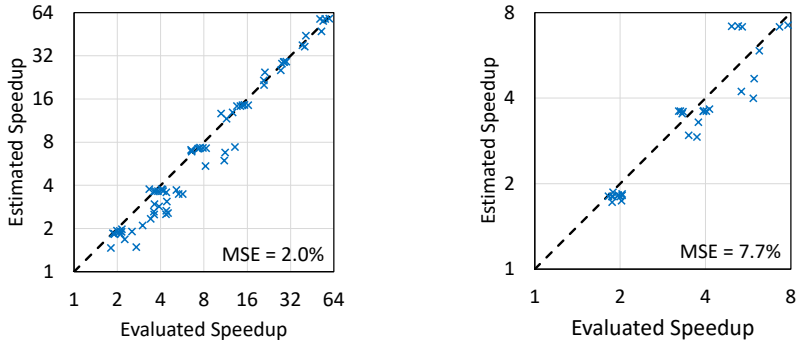
Regarding the scaling of model size, when the model size is relatively small (e.g., 125M), the communication time accounts for approximately 41.4% of the training time on Polaris with lower bandwidth; in contrast, it only takes 6.7% on TG40 in a high bandwidth environment. However, as the model size increases, communication gradually dominates the overall time. For instance, as illustrated in Fig. 12, when the model size becomes 6.7B, the communication time takes up to 85.4% of the training time on Polaris and 69.0% on TG40, which indicates that a higher bandwidth brings an advantage over a lower one, particularly when the model size is larger.

Further, the ZeRO-2 evaluation results are depicted in Fig. 13a. In contrast to ZeRO-1, the Reduce communication does not scale well across multiple GPUs on Polaris, as discussed in Section 4.2. Consequently, this increases the communication ratio by 5-10% in a training iteration, and reduces the overall training scalability. In comparison, Fig. 13b exhibits overall better scaling on TG40, despite the Reduce operation efficiency being the same. Moreover, the Reduce communication time for TG40 is, on average, 7.16 times shorter than Polaris, due to an 8 times higher interconnection link bandwidth. As a result, the communication contributes less to the overall training, and thus, the TG40 iteration time is 62.29% faster than Polaris, on average.

Moreover, the ZeRO-3 evaluation results, depicted in Fig.14a, reveal that ReduceScatter and AllGather also scale almost linearly, leading to a near-linear training scalability. In comparison, Fig. 14b plots the evaluation results on TG40. The communication time distribution is significantly reduced compared to Polaris. Overall, a high system bandwidth provides a tangible edge in communication scaling. However, achieving near-linear speedup is still possible regardless of whether the bandwidth is low or high.

Among all the ZeRO stages, as the model parameters increase, the communication volume also increases, leading to a higher communication distribution during a training iteration. Consequently, this reduces the non-overlapping computation time and makes the overall training throughput further bottlenecked by the network bandwidth. Regarding the comparisons between different ZeRO stages, if we increase the ZeRO stages while keeping the same number of GPUs and the same model size, more data will be sharded across multiple GPUs. Therefore, with a higher ZeRO stage, the system is able to accommodate a larger model, such as the 13B and 18.4B models, without encountering any OOM issues.

On the other hand, Table 7 shows the breakdown of per-iteration training time on all three systems, each equipped with 8 GPUs and utilizing ZeRO-1 optimization for the GPT and BERT



(a) Polaris. (b) TG40.

Fig. 15. Estimated speedup compared to evaluated speedup.

models. In general, the GPT and BERT models yield similar training times since the only difference between the two models is whether the multi-head attention is masked, as discussed in Section 2.1. Under the same model size, the compute time is similar across all three systems since the compute amount is the same. Furthermore, the AllReduce time is approximately twice that of the AllGather time, as previously mentioned in Section 2.4. Moreover, TG40 and TG80 provide similar speedups when compared to Polaris, due to the same high network bandwidth.

To assess the accuracy of our speedup analysis, we report the mean squared error (MSE) between the evaluated speedup and our predictions based on Equation (4). The evaluations are presented in Fig. 15. Our analysis, which takes into account both compute and communication, not only aligns with the evaluations in the same order of magnitude but also achieves precise per-iteration time estimation. It also indicates the effectiveness of our analytical model and our ability to accurately estimate training throughput after system scaling for various model configurations. Based on our evaluations, we emphasize the difference in communication time ratio between a lower bandwidth system (Polaris) and a high-bandwidth DGX machine (TG40), which leads to a different training landscape. The average time per training iteration on Polaris is 2.45 times longer than that on TG40, as the average communication time on TG40 is 7.35 times faster than that on Polaris. The average communication time distribution of a training iteration on TG40 is 26.70%, whereas it is 69.02% on Polaris, indicating a 42.32% increase in the communication time distribution.

4.4 Model Parallelism

In this section, we discuss the synergy between model parallelism and data parallelism. Fig. 16 and Fig. 17 show the ZeRO-1 stage evaluation with different model parallelism configurations on an 8-node Polaris system. The global and micro-batch sizes are fixed at 256 and 8, respectively. Additionally, we disable one or two (all) InfiniBand NICs to investigate the impact of limited bandwidth, resulting in three network bandwidth conditions: P2 with 25 GB/s, P1 with 12.5 GB/s, and P0 with around 5 GB/s. In the case of all InfiniBand NICs being disabled (P0), the communication will only utilize the plain TCP transport without any RDMA support.

Fig. 16 shows that, when we keep the degree of tensor parallelism as 1, as the degree of pipeline parallelism increases, the degree of data parallelism decreases, and the total communication time first reduces but then increases. This is because, the time taken by the AllGather operation used in data parallelism reduces, and the SendRecv time increases due to the increase in pipeline stages. Furthermore, if the degree of pipeline parallelism is 16, each node needs to process 2 pipeline stages

since there are only 8 nodes in total. As a result, if the degree of pipeline parallelism is greater or equal to the number of nodes (in this case, it is 8), data parallelism can be done within one node and can thus take advantage of higher intra-node bandwidth, thereby significantly reducing the AllGather time (by 6 times).

Similar observations can be made in bandwidth-limited environments as well. As shown in Fig. 16, the training time with the P0 setting drastically reduces when the degree of pipeline parallelism increases from 4 to 8. The reason for this is that, an inter-node AllGather needs to be performed when the degree of data parallelism is 8, whereas it becomes intra-node communication with higher bandwidth when the degree of data parallelism is 4. On the contrary, if the degree of pipeline parallelism exceeds the number of nodes, each node needs to process more than one pipeline stage, resulting in an increase of pipeline stages and bubbles that reduce training throughput. This highlights the significance of selecting an appropriate degree of parallelism. For instance, an optimal pipeline parallelism plan with a degree of 4 in P1, which corresponds to the bandwidth-limited case, yields approximately the same per-iteration training time as the non-optimal pipeline parallelism of 8 in the full-bandwidth P0 case.

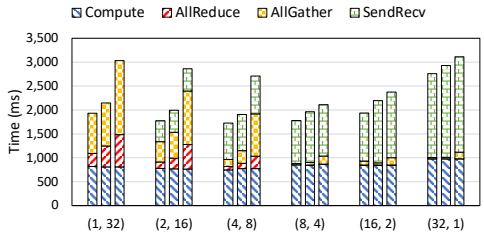
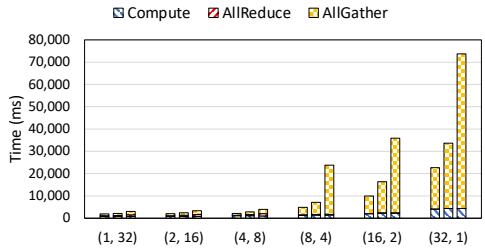
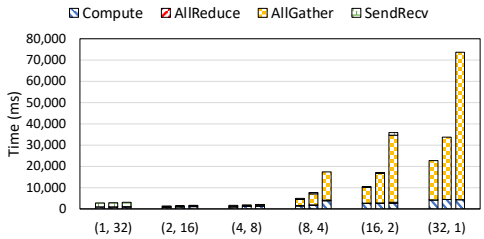


Fig. 16. (#PP, #DP) on 8-node Polaris. In each group, the training times per iteration are ordered as P2, P1, and P0.



(a) (#TP, #DP).



(b) (#TP, #PP).

Fig. 17. Model parallelism on 8-node Polaris under three network bandwidths with two (P2: 25 GB/s), one (P1: 12.5 GB/s), or none (P0: 5 GB/s) Infiniband enabled. In each group, the training times per iteration are ordered as P2, P1, and P0.

On the other hand, Fig. 17a shows that, as we increase the degree of tensor parallelism, the communication time increases, especially from 4 to 8. Since the activation memory will first be partitioned within a node in the tensor parallelism scheme, once the degree of tensor parallelism exceeds the number of GPUs per node (e.g., 4), the communication time will increase rapidly, given that the inter-node Infiniband bandwidth is much lower than the intra-node NVLink bandwidth. Comparing the results with the full-bandwidth settings P2, this issue becomes much more severe with the P0 and P1 settings. The communication time increases more drastically when the degree of tensor parallelism exceeds 4.

Additionally, the results shown in Fig. 17b align with Fig. 17a and Fig. 16. It indicates that when the degree of tensor and pipeline parallelism is set to 4 and 8, respectively, we can achieve the best speedup. A better speedup can also be achieved when the degree of tensor parallelism is smaller than the number of GPUs per node. In general, to achieve the best training throughput, we

would recommend setting the degree of tensor parallelism to match the number of GPUs per node and keeping the degree of pipeline parallelism smaller than the number of nodes, especially in a bandwidth-limited environment.

4.5 Discussion

In this section, we discuss our overall insights based on prior analysis and evaluations.

4.5.1 The advantages of bottom-up breakdown and analysis. In large-scale training systems, it is important to understand how the efficiency of each component changes as the system scales. This allows us to identify potential bottlenecks and develop scaling and parallelization strategies to achieve the best training throughput. The bottom-up time breakdown proposed in this work offers several advantages – (1) it provides information on inter-op and intra-op dispatch times, which are unavoidable and can consume up to 95% of the compute time in smaller model training. Prior works, which do not emphasize these overheads and primarily focus on FLOPS calculation, would likely fail to predict and match the scaling of smaller model training on large-scale systems; (2) while the FLOPS metric can provide information on overall hardware utilization, it is insufficient for analyzing the hybrid parallelism characteristics. It is challenging to understand the contribution of each parallel strategy to training throughput using FLOPS as a metric. The bottom-up time breakdown makes it clearer and easier to determine the impact of each parallelism strategy on each component of training time; and (3) when upgrading existing systems or designing new hardware for computation or communication, our bottom-up profiling offers a general method to analyze the scaling effect and accurately predict training throughput. In summary, a bottom-up breakdown and analysis could help users better understand the characteristics of their system and workload.

4.5.2 Network efficiency and overall throughput. In our evaluations, network efficiency would significantly affect the scaling of the system as well as the final training throughput. On the other hand, once we have information about network efficiency, we can estimate how well the scaling could be. Therefore, for future scaling analyses on large-scale systems, it is recommended to first obtain a breakdown of single-node communication and compute times, along with network efficiency through benchmarking. With this information, we can estimate end-to-end training times after system scaling. Moreover, this approach also applies to the scaling of transformer models, as the number of parameters, computation, and memory consumption have been thoroughly investigated. In a bandwidth-limited environment, network efficiency remains a critical indicator for multi-node scaling, and low-precision compression can help reduce the volume of communication. Furthermore, we can model network bandwidth from the perspective of network efficiency in cases where network is a shared resource among multiple tenants, such as in supercomputing systems or HPC in cloud services, or in a network with different transport options, such as Infiniband or Gigabit Ethernet. This suggests that network efficiency is a general and effective indicator that should always be considered when scaling a system.

4.5.3 Data and model parallelism strategies. In large transformer model training, data parallelism can achieve better scaling compared to tensor parallelism. This is because data parallelism reduces the gradient of weights, while tensor parallelism reduces the gradient of activations, which are typically much larger than the weights. To further reduce the communication volume, a lower ZeRO stage with less data sharding is preferred if the model is able to fit into the GPU memory.

For model parallelism, we conclude that the degree of pipeline parallelism should be larger than the number of nodes, while the degree of tensor parallelism should not exceed the number of GPUs per node. To further improve GPU utilization, we can choose the maximum micro-batch size that can fit into a single GPU memory, and the corresponding global batch size can be determined.

Moreover, in a bandwidth-limited environment, the best parallelism strategy is that the degree of pipeline parallelism equals to the number of nodes, and the degree of tensor parallelism equals to the number of GPUs per node, as the inter-node bandwidth is much lower than the intra-node bandwidth, and it will cause a dramatic increase in communication time. We would like to emphasize that such insights can only be gained through our bottom-up breakdown analysis.

Specifically, in pipeline parallelism, the per iteration time consists of communication time and pipeline bubbles. In particular, the communication time depends on the bandwidth, while the pipeline bubbles typically do not, as each pipeline stage is usually assumed to complete simultaneously. As a result, under the same degree of pipeline parallelism, if we continue to increase the inter-node bandwidth, the bubble time cannot be eliminated, even though the communication time could be reduced. Consequently, the total time is dominated by the bubble time as the bandwidth increases, and thus data parallelism becomes a better option – compared to pipeline parallelism – for scaling. In general, we can determine this boundary by calculating the communication cost of data parallelism compared to the overhead caused by the bubble in pipeline parallelism.

4.5.4 Guidelines for the future design and optimization of large-scale systems. Based on our analysis and evaluations, we provide the following guidelines for the future design and optimization of large-scale systems – (1) while hardware is continuously advancing to deliver increased computing power, it is crucial for people to also focus on network bandwidth in the context of large-scale systems. In the case of Large Language Models (LLMs) that heavily depend on parallelization across hundreds or thousands of nodes, inter-node bandwidth becomes a decisive factor in network efficiency and consequently has a significant impact on the final training throughput; (2) the interconnection link speed can vary by up to eight times across high-performance computing systems that have a similar order of FLOPS. To enhance the scalability of LLM training within existing systems, upgrading the interconnection speed should be the primary consideration; and (3) there are multiple ways to improve training throughput. Since the inter-op and intra-op dispatch times can introduce overheads, one approach is to optimize computation by either reducing operator dispatch time or dispatching PyTorch operators in parallel to conceal the dispatch time. Another strategy would be optimizing communication using in-network computing, such as smart NICs or NVIDIA SHARP switches [7]. Finally, achieving better overlap between computation and communication can also enhance throughput.

5 RELATED WORKS

In this section, we discuss related works in characterization and optimization of transformer model:

Characterizing Transformer Models. Several works proposed model parallelism optimization and analysis. [39] proposed tensor parallelism in transformer training and studied weak scaling within one node. In comparison, we study tensor parallelism across nodes, and based on our detailed communication breakdowns, we analyze the synergy of data and tensor parallelism. Further, [27] integrated pipeline parallelism to transformer model training and provided detailed analysis on pipeline bubbles. They also provided an end-to-end training throughput estimation based on FLOPS. However, their results were evaluated on high-bandwidth DGX servers, and thus, the estimation may not serve as a practical metric for a lower-bandwidth system. For instance, our evaluation showed that the communication time could be seven times longer, and the overall FLOPS could be reduced by 80%, thus deviating from the estimation using FLOPS. In general, our model parallelism characterization on high bandwidth settings aligns with their viewpoint, but we further break down the reasoning into the contribution from data and model parallelism. On the other hand, [23] focus on characterizing activation computation and still adopt FLOPS as a metric, and discussion and evaluation are made on high-bandwidth DGX servers.

Several works focus on communication optimization and analysis during transformer model training. [34] analyzed the minimum memory or disk bandwidth requirements to meet the arithmetic intensity for a given transformer model. In general, our analysis can further integrate with theirs to incorporate memory bandwidth. [17] reduced data movement during transformer model training by identifying operator dependency in compute-dataflow. Additionally, they proposed a memory efficiency metric, MUE, to quantify data movement performance instead of FLOPS.

On the other hand, parallelization and partitioning allow a larger transformer model to fit in a limited GPU memory during training. [41] conducted a comprehensive pipeline parallelism comparison and proposes a general automating graph partitioning, and [47] partitioned matrix multiplication in transformer layers to accommodate a larger size of activation memory and employs α-β network modeling for collective communication calls similar to the analysis in nccl-tests [4]. [31], on the other hand, focused on characterizing the GEMM operators in BERT models. In addition, multiple works [15, 20, 24, 32] provided guidelines on designing transformer model architecture and the data efficiency during training. However, most prior works focus primarily on analyzing FLOPS, which may not be representative of the end-to-end training time in a bandwidth-limited system. Therefore, the scaling law need to be adjusted to include bandwidth, and the resulting model scaling can be a shift from the existing compute-optimal scaling. We believe that taking into account both the computation and communication costs can make the scaling law more practical.

Transformer Model Training Optimization. The ZeRO optimization was first proposed and implemented in DeepSpeed [33, 36]. It was further integrated into PyTorch FSDP [49]. Subsequent DeepSpeed optimizations focused on memory offloading to CPUs or NVMe disks [34, 37]. Moreover, extensive research has been conducted on quantization for both training and inference to reduce memory consumption in previous works [44, 46, 48]. These prior works primarily concentrated on memory reduction and offloading for large models and were evaluated on high-end DGX servers.

On the other hand, Megatron-LM [23, 27, 39] integrated 3D parallelism, including data, tensor, and pipeline parallelism implementations. Megatron-DeepSpeed [3] further integrated ZeRO optimization with various model parallelism schemes within the Megatron-LM framework. Additionally, efficient 3D parallelism implementations parallel to Megatron-DeepSpeed are provided in [40, 43].

Meanwhile, deep learning compilers offer a systematic way to optimize transformer model computations. Many prior works [50, 51] on auto parallelism provided sharding schemes that balance compute and communication cost, but they primarily do not involve other optimizations such as ZeRO optimization. Additionally, several other works [2, 10, 18] concentrated on enhancing the performance of the NCCL communication library in terms of pipeline efficiency through compiler techniques.

6 CONCLUSION

This paper presents a thorough characterization and analysis of large transformer model training at scale. Our evaluation reveals that communication, on average, consumes 80% of the time during training on Polaris, and a lower bandwidth system deteriorates the communication time up to 22% compared to DGX systems. Furthermore, our bottom-up analysis, considering model configurations, network bandwidth, multi-core scaling and compute overheads, yields precise end-to-end training time estimations of MSE 2.0% across diverse model sizes and system architectures. While data parallelism is currently the prevalent strategy for large-scale transformer model training, we delve into the synergy between data parallelism and model parallelism by delineating the distinct roles of various communication calls, leading to training scaling under different network bandwidths. Driven by the ever-growing size of large transformer models, the future supercomputing landscape continues to evolve, and, more importantly, our study serves as a clarion call to characterize the often overlooked, yet pivotal role of network bandwidth during large-scale training. Through our

thorough characterization and analysis, the interplay between compute and communication during large-scale training is positioned to be foundational in driving efficient, scalable and performant system design, in an era revolutionized by ever-increasing model complexities.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous SIGMETRICS reviewers for their insightful comments and suggestions. This research was funded in part by and used resources at the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work is also supported in part by NSF grants #2008398, #1931531, and #2122155. Lastly, we would like to thank Meng-Yu Tsai for the intellectual discussions.

REFERENCES

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[2] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 62–75.

[3] Microsoft Corporation. 2022. Megatron-DeepSpeed. "https://github.com/microsoft/Megatron-DeepSpeed".

[4] Nvidia Corporation. 2016. NCCL Tests. "https://github.com/NVIDIA/nccl-tests".

[5] Nvidia Corporation. 2016. NVIDIA Collective Communications Library (NCCL). "https://github.com/NVIDIA/nccl".

[6] Nvidia Corporation. 2016. NVIDIA Nsight Systems. "https://developer.nvidia.com/nsight-systems".

[7] Nvidia Corporation. 2016. NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP). "https://docs.nvidia.com/networking/display/sharpv300".

[8] Nvidia Corporation. 2016. NVIDIA Tools Extension Library (NVTX). "https://github.com/NVIDIA/NVTX".

[9] Nvidia Corporation. 2023. CUDA Samples. "https://github.com/NVIDIA/cuda-samples".

[10] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2022. Gc3: An optimizing compiler for gpu collective communication. arXiv preprint arXiv:2201.11840 (2022).

[11] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in Neural Information Processing Systems 35 (2022), 16344–16359.

[12] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. 2023. Gpts are gpts: An early look at the labor market impact potential of large language models. arXiv preprint arXiv:2303.10130 (2023).

[13] GitHub. 2023. Copilot. "https://github.com/features/copilot".

[14] Hannibal046. 2023. Awesome-LLM. "https://github.com/Hannibal046/Awesome-LLM".

[15] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. An empirical analysis of compute-optimal large language model training. Advances in Neural Information Processing Systems 35 (2022), 30016–30030.

[16] Mikhail Isaev, Nic McDonald, and Richard Vuduc. 2023. Scaling Infrastructure to Support Multi-Trillion Parameter LLM Training. In Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023).

[17] Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2021. Data movement is all you need: A case study on optimizing transformers. Proceedings of Machine Learning and Systems 3 (2021), 711–732.

[18] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. 2022. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 402–416.

[19] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. Nature 596, 7873 (2021), 583–589.

[20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020).

[21] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT, Vol. 1. 2.

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).

[23] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. Proceedings of Machine Learning and Systems 5 (2023).

[24] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. arXiv preprint arXiv:2211.09110 (2022).

[25] Meta. 2022. PyTorch Profiler. "https://pytorch.org/docs/stable/profiler.html".

[26] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon. 2001. Top500 supercomputer sites. "https://www.top500.org/".

[27] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model

training on gpu clusters using megatron-lm. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 1–15.
[28] OpenAI. 2023. ChatGPT. "https://chat.openai.com".
[29] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.
[31] Suchita Pati, Shaizeen Aga, Nuwan Jayasena, and Matthew D Sinclair. 2021. Demystifying bert: Implications for accelerator design. arXiv preprint arXiv:2104.08335 (2021).
[32] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. arXiv preprint arXiv:2112.11446 (2021).
[33] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 1–16.
[34] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 1–14.
[35] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In International Conference on Machine Learning. PMLR, 8821–8831.
[36] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 3505–3506.
[37] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. {ZeRO-Offload}: Democratizing {Billion-Scale} model training. In 2021 USENIX Annual Technical Conference (USENIX ATC 21). 551–564.
[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 10684–10695.
[39] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053 (2019).
[40] Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. 2023. Optimus-CC: Efficient Large NLP Model Training with 3D Parallelism Aware Communication Compression. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 560–573.
[41] Masahiro Tanaka, Kenjiro Taura, Toshihiro Hanawa, and Kentaro Torisawa. 2021. Automatic graph partitioning for very large-scale deep learning. In 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 1004–1013.
[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Advances in neural information processing systems 30 (2017).
[43] Boxiang Wang, Qifan Xu, Zhengda Bian, and Yang You. 2022. Tesseract: Parallelize the tensor parallelism efficiently. In Proceedings of the 51st International Conference on Parallel Processing. 1–11.
[44] Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. 2023. ZeRO++: Extremely Efficient Collective Communication for Giant Model Training. arXiv preprint arXiv:2306.10209 (2023).
[45] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682 (2022).
[46] Xiaoxia Wu, Zhewei Yao, and Yuxiong He. 2023. ZeroQuant-FP: A Leap Forward in LLMs Post-Training W4A8 Quantization Using Floating-Point Formats. arXiv preprint arXiv:2307.09782 (2023).
[47] Qifan Xu and Yang You. 2023. An efficient 2d method for training super-large deep learning models. In 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 222–232.
[48] Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. 2023. ZeroQuant-V2: Exploring Post-training Quantization in LLMs from Comprehensive Study to Low Rank Compensation. arXiv preprint arXiv:2303.08302

(2023).

[49] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. 2023. Pytorch FSDP: experiences on scaling fully sharded data parallel. arXiv preprint arXiv:2304.11277 (2023).

[50] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. 2022. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 559–578.

[51] Yonghao Zhuang, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, Hao Zhang, and Hexu Zhao. 2023. On optimizing the communication of model parallelism. Proceedings of Machine Learning and Systems 5 (2023).

[52] Maxim Zvyagin, Alexander Brace, Kyle Hippe, Yuntian Deng, Bin Zhang, Cindy Orozco Bohorquez, Austin Clyde, Bharat Kale, Danilo Perez-Rivera, Heng Ma, et al. 2022. GenSLMs: Genome-scale language models reveal SARS-CoV-2 evolutionary dynamics. bioRxiv (2022), 2022–10.

Received October 2023; revised January 2024; accepted January 2024