Performance Profile of Transformer Fine-Tuning in Multi-GPU Cloud Environments

Edmon Begoli, Seung-Hwan Lim, Sudarshan Srinivasan
Oak Ridge National Laboratory
1 Bethel Valley Road, Oak Ridge, TN 37830, USA

***Abstract*—The study presented here focuses on performance characteristics and trade-offs associated with running machine-learning tasks in multi-GPU environments on both on-site cloud computing resources and commercial cloud services (Azure). Specifically, this study examines these trade-offs by examining the performance of training and fine-tuning of transformer-based deep-learning (DL) networks on clinical notes and data, a task of critical importance in the medical domain. To this end, we perform DL-related experiments on the widely deployed NVIDIA V100 GPUs and on the newer A100 GPUs connected via NVLink or PCIe. This study analyzes the execution time of major operations to train DL models and investigate popular options to optimize each of them. We examine and present the findings on the impacts that various operations (e.g. data loading into GPUs, training, fine-tuning), optimizations, and system configurations (single vs. multi-GPU, NVLink vs. PCIe) have on the overall training performance.**

I. Introduction

The research efforts presented here represent a collaboration between Oak Ridge National Laboratory and the US Department of Veterans Affairs (VA) to study and understand the performance characteristics and trade-offs associated with running machine-learning tasks using on-site cloud computing resources vs. commercial cloud services. Specifically, we are interested in training and fine-tuning a transformer-based deep-learning (DL) network on clinical notes, a task of critical importance. The VA curates one of the largest global repositories of clinical notes, including mental health records. To process these notes efficiently and accurately, our team investigated which configuration offers the best performance for DL problems and natural language processing (NLP) tasks.

Given the size and complexity of the data, and to best inform the VA's decisions, we used the VA's *BlueRidge* on-site high-performance computing (HPC) platform, and we compared the performance with a similar cloud-based solution. Specifically, we performed DL-related experiments on the widely deployed NVIDIA V100 GPUs and on the newer A100 GPUs in single and multi-GPU configurations connected via NVLink or PCIe.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive, paid up, irrevocable, world-wide license to publish or reproduce the published form of the manuscript, or allow others to do so, for U.S. Government purposes. The DOE will provide public access to these results in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

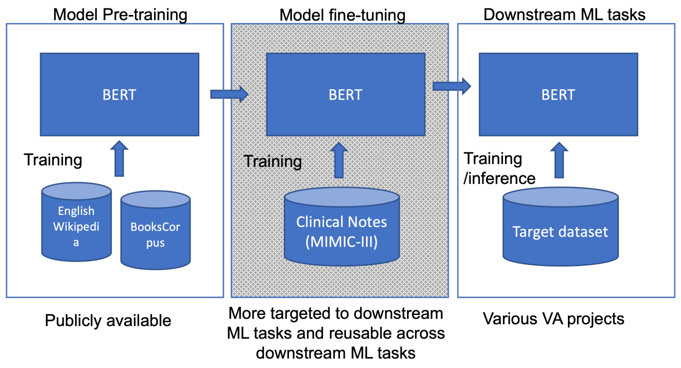

Fig. 1: Overview of fine-tuning a DL-based language model.

We then performed the same experiment on an equivalent configuration in the cloud. We also studied any potential impacts that the different I/O profiles of each platform might have on the DL training performance. We present the experimental configuration and results in the following sections.

II. Background

Language modeling is the task of learning the structure of natural language by using statistical and probabilistic techniques. This is accomplished by determining the probability of a given sequence of words occurring in a sentence. For this task, DL-based models, such as the Bidirectional Encoder Representations from Transformers (BERT) [1], demonstrated impressive results compared with prior methods. However, DL-based language models require enormous computing resources to train—from scratch—the large amount of text data required for a well-generalized model.

To overcome this challenge, language models are now being fine-tuned. This process involves models that are pretrained using generic text corpora, such as Wikipedia articles and the BookCorpus dataset, and then fine-tuning the model for a target text corpus. Figure 1 shows an overview of fine-tuning a DL-based language model. The fine-tuned language model specific to the target text is often used for downstream machine-learning tasks (e.g., classification of documents and other natural language–specific machine-learning tasks). Fine-tuning reduces the time required to train a language model and requires fewer computational resources. Fine-tuning also increases the prediction accuracy in the downstream machine-learning task.

Because this study specifically fine-tunes the BERT model for medical notes, we briefly introduce the BERT model and fine-tuning of language models, along with related work in fine-tuning with a focus on computational efficiency.

A. Transformer neural networks

The transformer architecture introduced by Vaswani et al. [2] is a sequence-to-sequence [3] model consisting of an encoder and a decoder. The encoder takes an input sequence and maps it into a higher-dimensional space. This abstract high-dimensional vector is then fed into the decoder, which then turns it into an output sequence.

Unlike other sequence-to-sequence models based on recurrent neural networks, the transformer architecture does not contain any recurrent connections. Transformer neural networks capture long-range dependencies using only the attention mechanism [4]. Attention enhances the important parts of the input data and fades out the rest. The transformer architecture also uses multi-headed attention and self-attention. Multi-headed attention helps capture dependencies of various ranges within a sequence. Self-attention helps capture contextual relationships between itself and other tokens.

B. Related work

A few efforts relevant to this study span two categories: performance benchmark studies of DL models and, more narrowly, fine-tuning large language models such as BERT.

1) Performance benchmark studies of DL models: Heterogeneous HPC systems with GPUs are equipped with high-performance interconnects such as InfiniBand, Omni-Path, PCIe, and NVLink. Multiple studies have captured the performance impact of these interconnects on distributed DL [5], [6]. Prior studies have discussed the impact on the end-to-end training time for different communication interconnects and identified the major bottleneck as the `allreduce` calls used for synchronization and gradient communication in updating the model parameters [6]. However, most prior studies of communication interconnects considered relatively well-known convolutional neural network models with imagery data [6]. More recently, a large language model (transformer) included in the MLPerf benchmark suite was studied [5]. However, this study was limited to a single-GPU system without the need for an interconnect, and the results do not consider the effects of communication interconnects on the training time. Because the behavior of DL workloads varies according to the input data characteristics and the model sizes, a large-scale language model requires a separate benchmark study. More specifically, because distributed DL workloads are communication bound in the model updates, we focus on the impact of communication interconnects on the training time for large language models such as BERT.

2) Performance benchmark studies of language model fine-tuning: Yihui Ren et al. [7] examined the performance of leading-edge systems designed for machine-learning computing, including the NVIDIA DGX-2, Amazon Web Services (AWS) P3, IBM Power System Accelerated Compute Server AC922, and a consumer-grade Exxact TensorEX TS4 GPU server. Representative DL workloads used in computer vision and NLP were the focus of the analysis. Performance optimization studies related to the NVIDIA Collective Communication Library (NCCL)/NVLink [8], [9] are of interest because they are critical in training DL models in distributed, multi-GPU environments. Our study also shows that the time spent optimizing models in multi-GPU environments varies significantly depending on the use of NVLink in communication between GPUs.

TABLE I: Hardware configurations: BlueRidge is a virtual machine (VM) running on VMware

System name	**GPU model**	**NVLink**
IDK	4× NVIDIA Tesla V100 SXM 16 GB	Yes
BlueRidge (VM)	3× NVIDIA GRID V100S 32 GB	No
Azure (NC24s_s)	4× NVIDIA Tesla V100-PCIE 16 GB	No
System name	**Storage system (file system/device)**	
IDK	NFS/Flash	
BlueRidge	NFS/Flash	
Azure (NC24s_s)	XFS/Premier SSD	
System name	**CPU**	
IDK	Intel(R) Xeon(R) Gold 6130 CPU @ 2.10 GHz	64 cores
BlueRidge (VM)	Intel(R) Xeon(R) Gold 6130 CPU @ 2.10 GHz	8 vCPUs
Azure (NC24s_s)	Intel(R) Xeon(R) E5-2690v4 CPU @ 2.60 GHz	24 cores
System name	**Memory**	
IDK	376 GB	
BlueRidge (VM)	32 GB	
Azure (NC24s_s)	440 GB	

III. System Configuration

Table I summarizes the hardware configurations of the test bed platforms used in our experiments. Among those test beds, the BlueRidge on-site cloud environment is described here in additional detail. BlueRidge includes a mix of high core count, high memory, and GPU-equipped environments. Flexible virtual allocations ensure that the underlying hardware is utilized at high efficiency while meeting the needs of users who need GPU or CPU-only resources.

GPUs are well suited for highly parallelizable ML/AI operations, and this project explores ways to maximize fixed-resource GPU use in on-site environments by providing a mix of dedicated GPUs for large training jobs and on-demand, virtualized GPUs for smaller training and inference jobs. BlueRidge can also dynamically schedule smaller resource pools of higher value, such as dedicated NVLink- or NVSwitch-connected GPUs for large training operations that require the higher bandwidth and larger aggregate GPU memory. However, we have not deployed NVLink-based GPUs in our BlueRidge environment. Instead, we deployed PCIe-based GPUs. The GPU configurations and allocation models are illustrated in Figure 2.

IV. Approach

To benchmark the specific workload studied here, we download a pretrained BERT model [1], known as *bert-base-uncased*, provided by the Hugging Face transformers

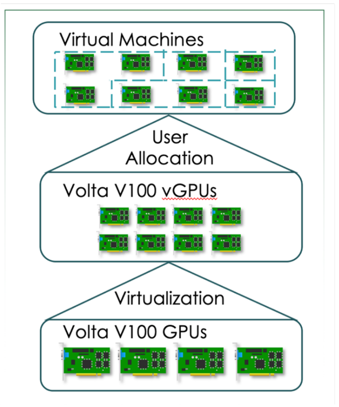

Fig. 2: GPU configuration and allocation model on BlueRidge.

library [10]. We then fine-tune this language model using the clinical notes provided in the Medical Information Mart for Intensive Care version 3 (MIMIC-III) dataset [11]. We use 90% of the notes as our training data and the remaining 10% as our testing data. We train the model for 3 epochs following standard practice for fine-tuning language models [12].

MIMIC-III is a large, single-center database comprising information relating to patients admitted to critical care units at Beth Israel Deaconess Medical Center in Boston, Massachusetts. It contains data associated with 58,976 distinct hospital encounters for 46,520 patients admitted to critical care units between 2001 and 2012. The dataset is stored in a relational database consisting of 26 tables with information regarding admissions, discharges, patients, procedures, prescriptions, and diagnoses, which are organized by Johnson et al. [11]. We are interested in the clinical notes that are part of this dataset. These notes are grouped into categories written by nurses, physicians, and other healthcare personnel and stored in the table called `NOTEEVENTS`. For training the language model, we use all of the nodes except those marked as erroneous.

A. *Performance under evaluation*

Figure 3 depicts the training process for a DL model, such as transformers, to train a batch. One epoch consists of a repetition of batches as the entire training dataset is iterated. We repeat the epoch multiple times until the DL model is optimized at the desired level. In a batch, our profiling results identified four major operations as organized Eliad et al. [13]: `get training batch`, `forward` (iteration), `backward` (iteration), and `optimizer` (or model update). These four operations consumed the majority of computation time and resources when training our case, which is consistent with other DL model training. We focus on these four operations in our performance evaluation.

B. *Experimental setup*

We employ the PyTorch profiler from PyTorch Lightning, which can leverage distributed data parallelism to measure GPU and CPU usage and record more detailed GPU usage

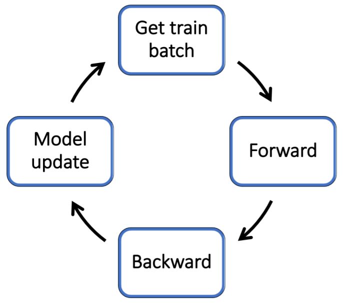

Fig. 3: Overview of the cyclical training process for a DL model, including batch loading, forward and backward iterations to identify correct network weights, and model updates.

TABLE II: Training model description (BERT for masked language model)

Parameter types	**Size**
Trainable parameters	109 MB
Non-trainable parameters	0
Total parameters	109 MB
Estimated modal parameter size	438.058 MB
Train batch size	4
Number of batches	190,121
Max sequence length	512

information for each operation during the model training. For training the language model, we use the transformer package from Hugging Face, which provides an interface to download pretrained models from the online repository and perform the required preprocessing operations, such as tokenizing the target text data. Table II summarizes important parameters of the pretrained model, and Table III summarizes the software configurations. For an environment behind a firewall or an offline system, such as BlueRidge in this study, we must ingest the data online prior to the execution.

C. *Experiments*

For all experiments, we used automatic mixed precision (`opt_level="O1"`) [14], which is recommended by NVIDIA for typical use to leverage the V100's Tensor Cores. Using level O1 automatic mixed precision casts inputs to all PyTorch functions and tensor methods according to a whitelist-blacklist model. Whitelist operations, such as general matrix multiply (GEMM) and convolutions, are performed in half-precision floating point (FP16). Blacklist operations that benefit from full-precision floating point (FP32), such as softmax, are performed in FP32. Level O1 also uses dynamic

TABLE III: System software configuration

Package name	**Version**
PyTorch Lightning	1.3.8
PyTorch	1.8.1
Hugging Face transformers	4.4.2
cuDNN	8.0.5
CUDA	11.2 (IDK) 11.4 (Azure) 10.2 (BlueRidge)

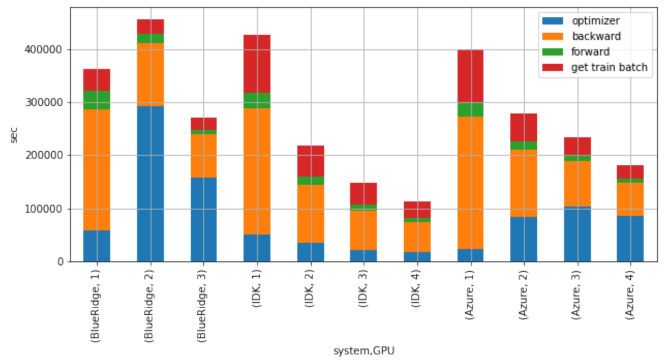

Fig. 4: Systems with NVLink (IDK) show better scalability as the number of GPUs increases compared with PCIe-based systems (BlueRidge and Azure). In PCIe-based systems, the time spent on `optimizer` (or model update) often increases as the number of GPUs increases.

loss scaling unless overridden. For communication between GPUs, we used the distributed data parallel mode provided by PyTorch Lightning (natively supported by PyTorch).

V. Results

A. Breakdown of the total execution time per operation

Figure 4 shows the breakdown of each major computation for each category during the execution of the considered application. Across systems, the total execution time (i.e., the height of the stacked bars) reduces as the number of GPUs increases, except in the BlueRidge-2 GPU case. Yet, for the BlueRidge-2 GPU case, the time spent on three major operations, `get train batch`, `forward`, and `backward`, does reduce as the number of GPUs increases. However, the time spent on the `optimizer` operation (blue region in each stacked bar) varies significantly depending on the system. To ascertain the root cause of this behavior, we look closer.

Because the optimizer operation involves the exchange of model parameters across GPUs, it is a communication-bound and CPU-bound operation. Thus, it is an operation that may not have a direct benefit from using multiple GPUs. Instead, it more or less can be seen as a challenge to maintain the benefits of using multiple GPUs in training DL workloads.

In this analysis, we see that a system with NVLink is more scalable than the others. We attribute this to the wider bandwidth that NVLink provides (40 GB/s) over PCIe (10 GB/s). As a result, the system with NVLink spent significantly less time on the `optimizer` operation than others, leading to the fastest and most scalable computation among the systems benchmarked here. Other PCIe-connected systems spent more time on `optimizer` as the number of GPUs increased, which meant more time spent on `optimizer` compared with other operations for each case.

The single-GPU case in BlueRidge ran faster than the single-GPU case in IDK, showing the importance of communication hardware (NVLink) and memory on the CPU side. BlueRidge was configured with 32 GB of RAM for the VM, which is the primary difference between the BlueRidge and Azure VMs. We speculate that memory size explains why performance degrades more seriously when two GPUs were used compared with one GPU in BlueRidge. The importance of communication hardware and system memory is a key lesson learned from benchmarking scalable AI applications.

Another interesting result among single-GPU cases is that BlueRidge performed better than other cases, though it ran in a VM on a VMware hypervisor. Compared with BlueRidge, the single-GPU Azure case spent $2.67\times$ more time on the `get train batch` operation, which offset the benefit of the $2.46\times$ faster optimization from having $13.7\times$ more system memory. As a result, the Azure singe-GPU case trained the same workload 15.6% slower than the BlueRidge single-GPU case. Because the `get train batch` operation is mostly loading training data from the storage system, this case shows differences between commercial cloud (Azure) and private cloud (BlueRidge) I/O patterns for DL workloads. We used the *Premier SSD* option in Azure, which is a high-end storage service that Azure provides.

Next, we consider options to reduce the time spent on two non compute–intensive operations: `get train batch` and `optimizer`.

B. Breakdown of the total execution time with an optimized get train batch operation

To optimize `get train batch`, we varied the number of data loaders/workers. The results are shown in Figure 5. By default, the number of data loaders/workers is 1, and they run within the same process as the model training. When we increase the number of loaders, PyTorch creates the matching number of threads to load the training data. The optimal number of loaders may depend on the size of the batch. In our case, the batch size is 4, which is the largest batch size that does not generate out-of-memory errors in the GPUs. We vary the number of data loaders between 1, 2, and 4. When we use 4 loaders, the efficiency plateaus, but with 2 loaders, the time spent on `get train batch` is less than 5% of the time spent with the default setting. Interestingly, the system with NVLink (IDK) shows a marginal but negative impact on compute-intensive operations, such as forward and backward operations, as the number of data loaders increases. Otherwise, the increased number of data loaders does not noticeably slow down compute-intensive operations. In short, we highly recommend increasing the number of data loaders if system memory capacity allows for it.

C. Breakdown of the total execution time with different NCCL parameters

NCCL offers parameters to control the communication behavior at the expense of more CPU utilization and more system memory usage:

- `NCCL_NSOCKS_PERTHREAD` specifies the number of sockets opened by each socket transport helper thread. For environments in which per-socket speed is limited,

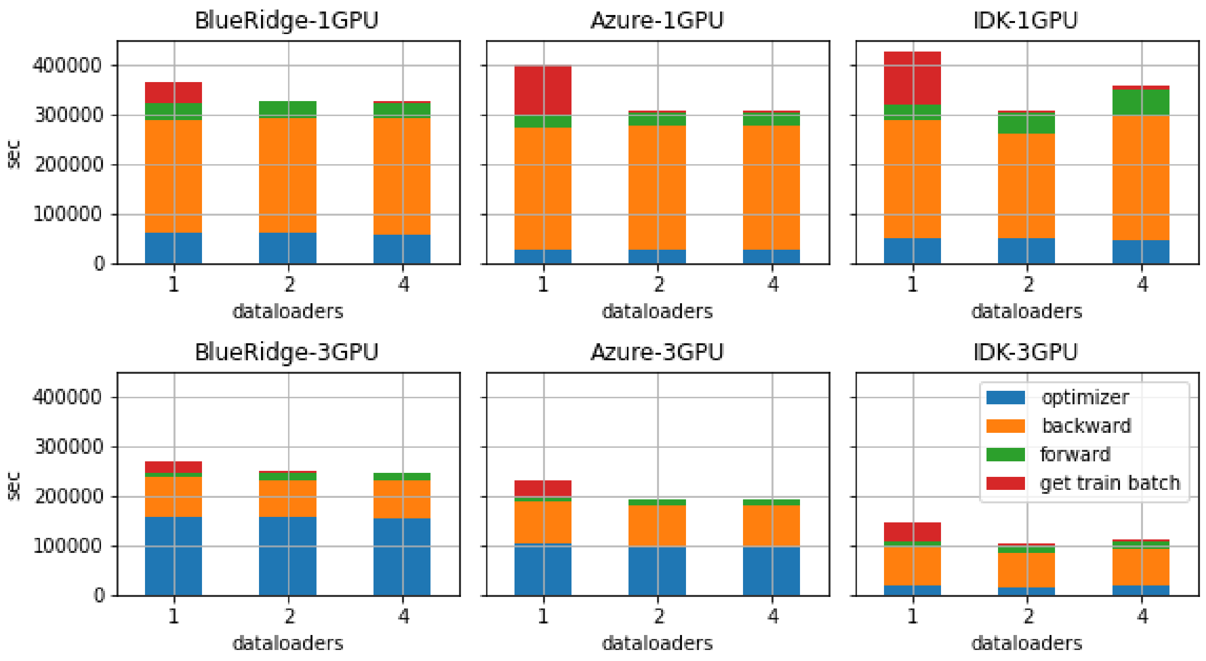

Fig. 5: The time required to load training data can be reduced by almost 5% by using multiple data loaders.

setting this variable larger than 1 may improve network performance.
The default value is 8 on AWS and 1 in other cases. For generic 100 Gbps networks, this value can be manually set to 4. However, the product of `NCCL_SOCKET_NTHREADS` and `NCCL_NSOCKS_PERTHREAD` cannot exceed 64.

- `NCCL_SOCKET_NTHREADS` specifies the number of CPU helper threads used per network connection for socket transport. Increasing this value may increase the socket transport performance at the cost of higher CPU usage. This parameter can be set from 1 to 16.
The default value is 2 on AWS, 4 on Google Cloud instances with the gVNIC network interface (since 2.5.6), and 1 in other cases.

Because Azure GPU instances, IDK, and BlueRidge do not set these parameters by default, we investigate the performance implications on training the BERT language model according to the above NCCL parameters. As shown in Figure 6, we confirm that those NCCL parameters do not exhibit performance benefits in single-node, multi-GPU environments. Based on this result, we speculate that NCCL parameters might be critical in distributed environments, rather than in single-node, multi-GPU environments.

VI. Summary

Based on our findings from the benchmarking and analysis, we offer several key insights:

- Loading training data and backward operations can be dominant operations in single-GPU environments. However, their total execution times reduce as we increase the number of GPUs. To further shorten the time to load training data, we recommend using 2–4 data loaders if the system memory capacity allows for it.
- When using multiple GPUs, `optimizer` performance becomes a bigger factor in PCIe-based systems. In PCIe-based multi-GPU systems, we recommend provisioning sufficient system memory to prevent serious slowdown of the `optimizer` operation. Because NCCL parameters are not as useful in single-node, multi-GPU systems, there are diminishing returns when using many GPUs in PCIe-based systems.
- NVLink outperforms PCIe in single-node, multi-GPU environments.

In summary, NVLink is a crucial component in scalable multi-GPU environments, and research related to reducing the communication overheads in the `optimizer` operation will be important efforts toward scalable training of large language models with large text datasets.

VII. Conclusions

In this study, we presented the implication of training time affected by the hardware configuration of systems in both commercial and on-site cloud environments, with a focus on NVLink vs. PCIe for GPU communication. We also investigated the possible options for optimizing major operations in training DL models and used fine-tuning of the BERT language model as an example. According to our experiments, systems that connect GPUs over PCIe show less effective scalability than systems with NVLink owing to the inefficiency of the communication-intense model optimization operation. In addition, the use of multiple data loaders was

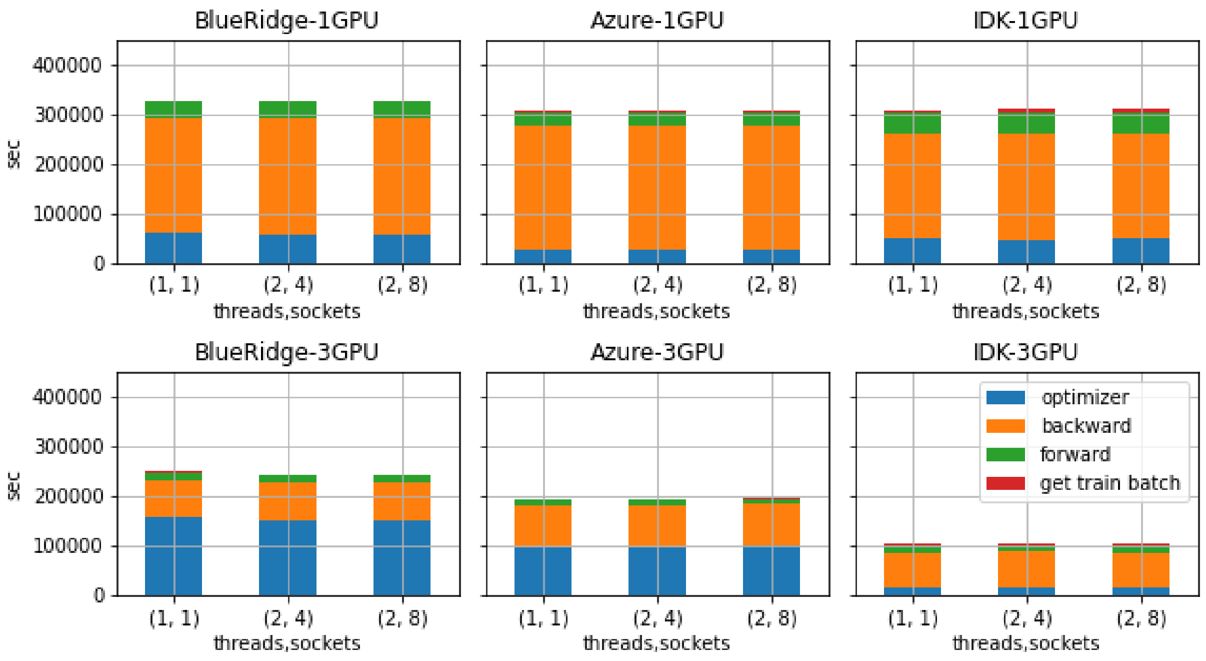

Fig. 6: NCCL parameters do not have an impact on the execution time for training the BERT model. Two data loaders were used in this case.

very effective in optimizing the time to load training data across all tested systems. Tuning NCCL parameters, another popular optimization technique, showed no clear benefits for communication operations across the tested systems. We attributed this behavior to the tested systems all being single-node, multi-GPU systems. We hope that this study will be informative to both system and algorithmic optimizations in reducing scalability bottlenecks when training large language models.

References

[1] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.
[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
[3] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
[5] Snehil Verma, Qinzhe Wu, Bagus Hanindhito, Gunjan Jha, Eugene B John, Ramesh Radhakrishnan, and Lizy K John. Demystifying the mlperf training benchmark suite. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 24–33. IEEE, 2020.
[6] Ammar Ahmad Awan, Arpan Jain, Ching-Hsiang Chu, Hari Subramoni, and Dhableswar K Panda. Communication profiling and characterization of deep-learning workloads on clusters with high-performance interconnects. *IEEE Micro*, 40(1):35–43, 2019.
[7] Yihui Ren, Shinjae Yoo, and Adolfy Hoisie. Performance analysis of deep learning workloads on leading-edge systems. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 103–113. IEEE, 2019.
[8] Shaohuai Shi, Xianhao Zhou, Shutao Song, Xingyao Wang, Zilin Zhu, Xue Huang, Xinan Jiang, Feihu Zhou, Zhenyu Guo, Liqiang Xie, et al. Towards scalable distributed training of deep learning on public cloud clusters. *Proceedings of Machine Learning and Systems*, 3, 2021.
[9] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. Blink: Fast and generic collectives for distributed ml. *Proceedings of Machine Learning and Systems*, 2:172–186, 2020.
[10] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.
[11] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.
[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
[13] Saar Eliad, Ido Hakimi, Alon De Jagger, Mark Silberstein, and Assaf Schuster. Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism. In *2021 USENIX Annual Technical Conference (ATC 21)*, pages 381–396. USENIX Association, 2021.
[14] NVIDIA. Amp: Automatic mixed precision. https://nvidia.github.io/apex/amp.html#o1-mixed-precision-recommended-for-typical-use. Online: accessed 12 Aug 2021.