Distributed LLM Training in GPU Clouds: Methods and Opportunities

1 Introduction

Large language models (LLM) have revolutionized the field of natural language processing (NLP) and have garnered significant attention in academic and industrial communities. These models are based on deep learning techniques, specifically Transformer [1] neural networks, and are trained on massive amounts of text data to learn the patterns and relationships within language.

Training LLMs requires a cluster of accelerator devices such as GPUs and specialized distributed training systems. In this document, we examine the characteristics of LLM training and its impacts on the system design, and then discuss some of the state-of-the-art optimization methods and potential opportunities.

2 Background

2.1 Transformer Models

Transformer [1] layers are the backbone of most state-of-the-art models. The input tensor to a transformer layer has the shape (B, S, H), where B is the batch size, S is the sequence length (number of tokens in a sentence), and H is the embedding length. A Transformer layer is composed of a multi-head self-attention layer [1] followed by an MLP layer. The attention layer calculates an attention matrix across the tokens in a sequence. Without additional optimizations, this operation has quadratic time and space complexity with respect to S [2]. The MLP layer is applied to each token individually. Therefore, it has a linear time and space complexity with respect to S.

Transformer layers expect the input tokens to be feature vectors. Additional encoders are used to convert the raw inputs into feature vectors according to their modalities. For example, when processing a text input, word embedding layers are used to convert token id numbers into feature vectors. Word embedding layers are simple trainable matrices of the shape (D, H), where D is the dictionary size and H is the embedding length. When processing image or audio inputs, more complex encoders such as the image patch embedding layer [3] are used. Those encoders may have different resource requirements than the main transformer network. For instance, the word embedding layers require a large amount of memory but have nearly no computation, while the image patch embedding layer is composed of computation-intensive convolution operations.

To further scale transformer models, recent studies have been exploring mixture-of-expert (MoE) layers [4, 5, 6, 7, 8], which can enlarge the model capacity (number of parameters) without increasing the computation time. An MoE layer comprises a number of conditionally activated experts. A gating network is used to compute the scores of each expert for each token and route the tokens to k experts with the highest scores. MoE models usually scale the total number of experts proportionally to the number of devices while choosing a fixed value for k. To route tokens to their corresponding experts, `All-to-All` operations are used before and after the MoE layers [4].

2.2 Distributed Training Methods

Recent models are becoming increasingly large and do not fit into a single device. Many distributed training methods have been proposed to train DNN models on GPU clusters,

including data parallelism (DP), model parallelism (MP), pipeline parallelism [9, 10], etc. These methods can be applied to different parts of the model, resulting in a mixed parallelism; they can also be used together on the same part of the model to form the so-called 2D or 3D parallelism.

In transformer models, the MLP layer can be partitioned along any of the B, S, and H dimensions, resulting in data parallelism, sequence parallelism [11, 12], and intra-layer model parallelism, respectively. On the other hand, since the attention layer works on the sequence level and requires all tokens in a sequence, it cannot be easily partitioned on the S dimension.

All distributed training methods come with communication overheads. With data parallelism, the parameters of the model are replicated on all devices. Each device calculates the gradient of parameters regarding a distinct portion of the input data batch, and these gradients need to be aggregated across devices using the `All-Reduce` communication before being applied to the parameters. With model parallelism, the parameters are sharded on the H dimension and each device only stores a shard. `All-Gather` and `Reduce-Scatter` are usually used in MP. Pipeline parallelism splits the model into stages (consecutive layers) and places each stage on different devices. The input minibatch is also split into several micro batches. When device i is processing stage i of micro batch j, device $i+1$ can process stage $i+1$ of micro batch $j-1$. With the expert parallelism used in MoE models [4], tokens are routed to different experts with the `All-to-All` operation.

3 Communication Optimization

3.1 Holistic Model Partitioning Strategy

Existing DNN training frameworks, such as DeepSpeed [13], allow choosing the sharding strategy for each kind of operation in the network. For example, the user can choose to use expert parallelism for MoE layers and data parallelism for other layers. However, this does not always lead to optimal performance. Communication is required when the producer and consumer of a tensor are sharded in an incompatible way. For example, if tensor z is produced by an attention layer that is sharded along the B dimension and is consumed by an MLP layer that is sharded on the H dimension, an `All-to-All` operation is required to convert the sharding dimension of z. Therefore, the optimal sharding method for an operator does not only depend on the operator itself, but also on the sharding methods of its neighbors, necessitating a holistic partitioning strategy on the model level.

Dynamic programming [14, 15, 16], mixed integer linear programming [17], random search [18, 19], and reinforcement learning [20, 21] approaches have been explored to automatically generate partitioning strategies for a given model.

Since transformer models are usually identical stacks of transformer layers, expert-designed sharding strategies such as Megatron [22] tend to work well. However, these methods quickly become suboptimal when modifying the model structure, such as adding patch embedding layers for multi-modality support [3], adopting MoE layers [23], or using local attention [24]. In addition, expert-designed sharding strategies usually only consider homogeneous clusters with high inter-connection bandwidth.

Unity [19] exhibits a 3.6× training speed-up for models that do not have highly optimized partitioning strategies. When training on heterogeneous clusters, AccPar [25] brings up to 6.3× speed-up.

3.2 Gradient Compression

Gradient compression techniques reduce the communication volume of gradients with lossy compression. The most commonly used methods include quantization [26] and sparsification

[27]. Quantization encodes the elements (16-bit or 32-bit floating point numbers) in gradient tensors with fewer bits, such as 8-bit or even 1-bit. To reduce the quantization errors, the encoding residuals are saved and incorporated in the next round of communication. Sparsification discards elements whose absolute values are smaller than a threshold value, and then uses a more efficient encoding for the resulting sparse tensor. These two methods can be used together. State-of-the-art gradient compression techniques can bring up to 42.5% training speed-up [28].

Lossy [29] and lossless [30] low-rank decomposition reduce the communication volume by replacing a gradient matrix M with two smaller matrices P and Q that PQ^T equals or approximates M. They can reduce the training time by up to 55%, without significant impacts on the convergence and final model accuracy [29].

3.3 Heterogeneous Network

Common communication libraries, such as NCCL [31], are mostly optimized for homogeneous networks. In public clouds, the inter-machine connections are often heterogeneous. Some of the machines may co-locate on the same rack and have a stable connection with a high bandwidth, while some other machines that locate in another room may have a worse connection. Existing communication libraries cannot exploit the higher bandwidth between co-located machines and will be bottlenecked by the slowest link.

Topology-aware collective communication methods have been proposed to better utilize heterogeneous networks. Blink [32], PLink [33], and BlueConnect [34] are designed for hierarchical topologies. SCCL [35] and TACCL [36] propose automated methods to synthesize optimal collective communication for a given topology. These methods have exhibited superior performance to NCCL even on seemingly uniform topologies, with up to 17% end-to-end training speed-up for MoE models [36].

4 Computation Optimization

4.1 Optimized Kernel Implementation

Recent work [37] has shown that the attention operation in transformer models does not fully utilize the GPU even with sufficiently large batch size and sequence length, due to its frequent access to the global memory. More optimized implementations, such as Flash Attention [38], can speed up LLM training by up to 3 times.

Though highly optimized implementations are available for common operations, they cannot easily be composited or altered to implement new operations such as local attention [24] or sparse attention [39]. Automated methods, such as TVM [40] and Triton [41], are proposed to optimize new computation patterns.

4.2 Parallel Execution

Another way to increase GPU utilization is to schedule multiple operations in parallel, as illustrated by IOS [42] and FasterMoE [43]. Though there are few opportunities for parallel execution in regular transformer models, MoE layers and image patch embedding introduce operations that could benefit from this optimization. When training CNN models, IOS [42] outperforms commonly used libraries by 1.5×.

4.3 Kernel Fusion

Kernel fusion merges multiple GPU kernels into a single kernel to reuse SRAM or even registers and reduce the number of global GPU memory access. Existing DNN training

frameworks such as PyTorch JIT and DeepSpeed fuse kernels using a set of pre-defined rules. Recent work [44] shows that automated fusion methods can bring up to 26.7% speed up when training transformer models.

5 Computation and Communication Overlap

5.1 Computation and Communication Overlap with intra-layer model parallelism

Overlapping the `All-Reduce` communication with backward computation has been a standard practice for data parallelism [45, 46, 13]. However, the communication introduced by model parallelism cannot be easily overlapped with computation due to data dependency. HiDup [16], Merak [47], and Oases [48] propose to split a minibatch into micro batches and overlap the computation of one micro batch with the communication of another. Wang et al. [49] proposes decomposing computation operations to form local pipelines. When training in public clouds with limited bandwidths, HiDup [16] can achieve up to 61% training speed-up for MoE models.

5.2 Pipeline Planning and Scheduling

With pipeline parallelism, the communication of activations between devices naturally overlaps with the computation. However, the performance depends heavily on the stage division and scheduling. Current DNN training frameworks often use simple stage division methods that balance the number of layers on each stage and use the simple GPipe-style [10] scheduling, which has been shown to not achieve optimal performance. More sophisticated methods like SPP [50] and DAPPLE [51] can improve the overlapping ratio and better balance the workload among devices, with up to 3.23× end-to-end speed-up [51].

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Markus N Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[4] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[5] Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, et al. Gspmd: General and scalable parallelization for ml computation graphs. *arXiv preprint arXiv:2105.04663*, 2021.

[6] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[8] Junyang Lin, Rui Men, An Yang, Chang Zhou, Ming Ding, Yichang Zhang, Peng Wang, Ang Wang, Le Jiang, Xianyan Jia, et al. M6: A chinese multimodal pretrainer. *arXiv preprint arXiv:2103.00823*, 2021.

[9] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[10] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32:103–112, 2019.

[11] Zhengda Bian, Hongxin Liu, Boxiang Wang, Haichen Huang, Yongbin Li, Chuanrui Wang, Fan Cui, and Yang You. Colossal-ai: A unified deep learning system for large-scale parallel training. *arXiv preprint arXiv:2110.14883*, 2021.

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

[13] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 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*, pages 3505–3506, 2020.

[14] Minjie Wang, Chien-chin Huang, and Jinyang Li. Supporting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–17, 2019.

[15] Linghao Song, Jiachen Mao, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. Hypar: Towards hybrid parallelism for deep learning accelerator array. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 56–68. IEEE, 2019.

[16] Shiwei Zhang, Lansong Diao, Chuan Wu, Siyu Wang, and Wei Lin. Accelerating large-scale distributed neural network training with spmd parallelism. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 403–418, 2022.

[17] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Joseph E Gonzalez, et al. Alpa: Automating inter-and intra-operator parallelism for distributed deep learning. *arXiv preprint arXiv:2201.12023*, 2022.

[18] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *arXiv preprint arXiv:1807.05358*, 2018.

[19] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, et al. Unity: Accelerating {DNN} training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 267–284, 2022.

[20] Xiaodong Yi, Shiwei Zhang, Ziyue Luo, Guoping Long, Lansong Diao, Chuan Wu, Zhen Zheng, Jun Yang, and Wei Lin. Optimizing distributed training deployment in heterogeneous gpu clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 93–107, 2020.

[21] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, et al. Gdp: Generalized device placement for dataflow graphs. *arXiv preprint arXiv:1910.01578*, 2019.

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[23] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[24] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[25] Linghao Song, Fan Chen, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. Accpar: Tensor partitioning for heterogeneous deep learning accelerators. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 342–355. IEEE, 2020.

[26] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[27] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[28] Saurabh Agarwal, Hongyi Wang, Shivaram Venkataraman, and Dimitris Papailiopoulos. On the utility of gradient compression in distributed training systems. *Proceedings of Machine Learning and Systems*, 4:652–672, 2022.

[29] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[30] Pengtao Xie, Jin Kyu Kim, Yi Zhou, Qirong Ho, Abhimanu Kumar, Yaoliang Yu, and Eric Xing. Distributed machine learning via sufficient factor broadcasting. *arXiv preprint arXiv:1511.08486*, 2015.

[31] Sylvain Jeaugey. Nccl 2.0. In *GPU Technology Conference (GTC)*, volume 2, 2017.

[32] 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.

[33] Liang Luo, Peter West, Jacob Nelson, Arvind Krishnamurthy, and Luis Ceze. Plink: Discovering and exploiting locality for accelerated distributed training on the public cloud. *Proceedings of Machine Learning and Systems*, 2:82–97, 2020.

[34] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. *Proceedings of Machine Learning and Systems*, 1:241–251, 2019.

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

[36] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. Synthesizing collective communication algorithms for heterogeneous networks with taccl. *arXiv preprint arXiv:2111.04867*, 2021.

[37] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

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

[39] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[40] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.

[41] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

[42] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. Ios: Inter-operator scheduler for cnn acceleration. *Proceedings of Machine Learning and Systems*, 3:167–180, 2021.

[43] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 120–134, 2022.

[44] Xiaodong Yi, Shiwei Zhang, Lansong Diao, Chuan Wu, Zhen Zheng, Shiqing Fan, Siyu Wang, Jun Yang, and Wei Lin. Optimizing dnn compilation for distributed training with joint op and tensor fusion. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4694–4706, 2022.

[45] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 463–479, 2020.

[46] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[47] Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1466–1478, 2023.

[48] Shengwei Li, Zhiquan Lai, Yanqi Hao, Weijie Liu, Keshi Ge, Xiaoge Deng, Dongsheng Li, and Kai Lu. Automated tensor model parallelism with overlapped communication for efficient foundation model training. *arXiv preprint arXiv:2305.16121*, 2023.

[49] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, et al. Overlap communication with dependent computation via decomposition in large deep learning models. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 93–106, 2022.

[50] Ziyue Luo, Xiaodong Yi, Guoping Long, Shiqing Fan, Chuan Wu, Jun Yang, and Wei Lin. Efficient pipeline planning for expedited distributed dnn training. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 340–349. IEEE, 2022.

[51] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021.