Building connection between Megatron-LM and HuggingFace

Badri Narayanan Murali Krishnan
Shreyas Subramanian
Rohan Rajesh Purandhar
Nevin John Selby
bmuralikrish@wisc.edu, ss7@wisc.edu
purandhar@wisc.edu, nselby@wisc.edu

Github Link: https://github.com/MBadriNarayanan/CS744

1 Introduction

With the continued evolution of transformer language models, the expansion of parameter sizes has become imperative for enhancing their efficacy in handling natural language processing tasks. However, the creation of larger models encounters the limitations of contemporary hardware accelerators, particularly GPUs. Models like BERT and GPT-2, when scaled up to billions of parameters, are confronted with memory constraints on single devices, rendering training infeasible. This difficulty is exacerbated by traditional optimization algorithms, which limit model sizes because of the increased memory needs necessary to store momentum and other optimizer states effectively.

The key to overcoming these obstacles is to make training on various GPUs possible. The distribution of the computational load among multiple accelerators reduces the load on individual devices, enabling the smooth operation of training procedures for very large models. This method not only overcomes memory constraints but also uses GPU clusters' inherent parallel processing power to speed up training times and make it easier to explore increasingly intricate architectures and hyperparameter spaces. As such, the introduction of multi-GPU training not only broadens the scope of transformer models but also drives breakthroughs in natural language generation and understanding to previously unheard-of degrees of complexity and power.

Training very large models with billions of parameters can be quite difficult due to memory constraints on modern hardware accelerators like GPUs [1]. As models become larger, they exceed the memory limit of modern processors, requiring additional memory management techniques such as activation checkpointing. [2]. Widely used optimization algorithms such as ADAM require additional memory per parameter to store momentum and other optimizer states, which reduces the size of models that can be effectively trained. Approaches like GPipe [3] and Mesh-TensorFlow [4] provide frameworks for model parallelism, but require rewriting the model and using custom compilers.

2 Related Work

The landscape of training large language models has been marked by an array of challenges, prompting extensive exploration and innovation in the field. In navigating the complexities inherent to these endeavors, researchers have encountered various obstacles, each demanding tailored solutions for effective model development and optimization.

2.1 Pipeline Parallelism

The authors of [5] introduce fine-grained pipeline parallelism for auto-regressive models like GPT, effectively utilizing multiple GPUs by exposing parallelism across tokens in a single training sequence. This technique optimizes training efficiency for large language models by maximizing parallel computation within a single sequence. PipeTransformer [6] dynamically adjusts pipelining and data parallelism by freezing layers with stable weights, allowing resources to focus on training active layers. By optimizing the balance between pipeline and data parallelism, PipeTransformer enhances training efficiency for large models on multiple GPUs.

Pipedream [7] Integrates pipeline and data parallelism to minimize cross-device communication, enhancing training efficiency for large models on multiple GPUs. By optimizing parallelization strategies, PipeDream aims to reduce training time and improve scalability without compromising accuracy.

Table 1: Timeline for the project

Activity / Weeks	0	1	2	3	4	5	6	7	8
Project Introduction									
Environment Setup									
Base LLM Formulation & Training									
Base Model Results									
LLM Model Formulation using Megatron LM									
Project Check-In									
Fine-tune LLM Model using Multiple GPUs									
Tabulate & Visualize Results									
Final Project Report									

2.2 Data Parallelism and Optimization

ZeRO [8] posits sharded data parallelism, distributing optimizer state and weight parameters across data-parallel workers. By optimizing communication and computation, ZeRO improves training efficiency for large models on multiple GPUs while maintaining scalability.

ZeRO-Infinity [9] utilizes NVMe to efficiently swap parameters, enabling the training of very large models on a small number of GPUs. This approach optimizes memory usage and communication overhead, allowing practical training of extremely large models on limited hardware resources.

3 Proposed Work

Hugging Face [10] currently lacks native support for the Megatron-LM framework which enables efficient training of transformer-based language models across multiple GPUs [1].

Our goal in this work is to perform a thorough assessment of the fine-tuning performance of language models on various computational setups. To set a baseline performance, we will first optimize a large language model (LLM) on a single GPU. This baseline will be used as a point of comparison to evaluate the possible advantages of using multiple GPUs and model parallelism strategies.

Later, we will use Megatron-LM and Hugging Face to refine the same LLM on the same dataset. We can take advantage of Megatron-LM's model parallelism capabilities and spread the training process over several GPUs by using it. On the other hand, the Hugging Face method will depend on its integrated data parallelism techniques, which might restrict the largest model size because of memory limitations.

Following both approaches' fine-tuning of the LLM, we will perform a thorough analysis and visual aid for the outcomes. Different performance metrics, including training time, convergence rates, and model quality on downstream evaluation tasks, will be compared in this analysis. We hope to measure the advantages of model parallelism and pinpoint any potential drawbacks or restrictions by directly contrasting the results of single-GPU and multi-GPU configurations.

In the end, this project will advance knowledge of the trade-offs associated with optimizing large language models through the use of various parallelism techniques and computational resources.

4 Timeline and Evaluation Plan

Table 1 illustrates the timeline for our project. To evaluate our project we plan to do the following:

- The empirical performance between multiple GPU training and single GPU training will be measured using metrics such as precision, accuracy, recall, and F1-score.
- We will also compare the performance and memory usage of the two different training methods. Mainly, we will tabulate the values of CPU and GPU usage.
- We will also compare the training time for each batch and iteration for the two different training methods.

References

[1] 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.

[2] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. *arXiv preprint arXiv:1809.02839*, 2018.

[3] 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, 2019.

[4] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

[5] Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, Dawn Song, and Ion Stoica. Terapipe: Token-level pipeline parallelism for training large-scale language models. In *International Conference on Machine Learning*, pages 6543–6552. PMLR, 2021.

[6] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. Pipetransformer: Automated elastic pipelining for distributed training of transformers. *arXiv preprint arXiv:2102.03161*, 2021.

[7] 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.

[8] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[9] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 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*, pages 1–14, 2021.

[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.