Final Project Report - Group 10

Ruilin Jin
Department of Computer and Data Science
School of Engineering
rxj420@case.edu

Xiaoge Zhang
Department of Computer and Data Science
School of Engineering
xxz705@case.edu

Yimin Huang
Department of Computer and Data Science
School of Engineering
yxh849@case.edu

Abstract

This project evaluates a custom Fully Sharded Data Parallel (FSDP) framework using PyTorch and MPI4Py for training the GPT-2 and BERT model. The aim is to compare the efficiency and effectiveness of this custom framework against PyTorch Lightning's standard Distributed Data Parallel (DDP) and FSDP methods. We first establish a performance baseline using PyTorch Lightning's distributed training features. Then, we develop and test a novel FSDP system with MPI4Py, focusing on managing the extensive requirements of the GPT-2 model. Performance is measured in terms of training time, GPU memory usage, throughput, and model accuracy. Success is defined by the custom framework's ability to enhance training efficiency and scalability, compared to the baselines, while maintaining or improving model accuracy. The project aims to provide insights into the potential of custom FSDP implementations for large-scale model training.

1 Introduction

1.1 Objective

The project aims to create a simulation of an FSDP system using PyTorch for the GPT-2 and BERT model, to evaluate the efficacy and efficiency of custom distributed training. The performance of this system will be compared against the baseline provided by the FSDP implementations in PyTorch Lightning.

1.2 Background and Related Work

1.2.1 Choosing PyTorch over mpi4py for implementation. mpi4py provides Python bindings for the Message Passing Interface (MPI) standard, allowing Python applications to exploit multiple processors on workstations, clusters and supercomputers. Our group carefully examine the possibilities of combining large language deep learning models and mpi4py and decides it requires much work which is hard to be finished within the scope of this project. Firstly, there's few tutorials of combining mpi4py and PyTorch libraries, making it hard for our group to implement because most of the implementations for large language model are implemented in libraries such as PyTorch and TensorFlow, and it is not applicable for our group to implement large language model from scratch (not requiring any external libraries). Secondly, the mpi4py only provide the functionalities for the flows of the data among hardware or computing clusters, meaning that if we implement large language model using mpi4py, we will need to determine the flows of the data between the nodes and develop our own synchronization strategies. Within the scope of the course, it is difficult to achieve. Therefore, we decide to take some developed version of LLM and then use the distributed data parallelism library of PyTorch to modify them into the distributed version of the large language models.

1.2.2 PyTorch Lightning. PyTorch Lightning is a streamlined framework for deep learning that wraps around PyTorch, offering advanced tools to simplify model development. It comes with built-in support for popular distributed training strategies, including Distributed Data Parallel (DDP) and Fully Sharded Data Parallel (FSDP). DDP is a strategy where the model is replicated across multiple GPUs, with gradients being synchronized across these replicas. FSDP, on the other hand, divides the model's parameters across GPUs, allowing for the training of models larger than the memory capacity of a single GPU by reducing the memory requirements on each GPU.

1.2.3 GPT-2. GPT-2[4], launched by OpenAI in February 2019, represents a significant milestone in the field of natural language processing. This model is built on the transformative transformer architecture, which enables it to effectively manage complex language patterns and dependencies. With its 1.5 billion parameters, GPT-2 stands out for its ability to generate coherent and contextually relevant text. Its extensive training on a diverse range of internet-sourced data allows it to have a broad understanding of various topics and language nuances.

1.2.4 BERT. BERT (Bidirectional Encoder Representations from Transformers)[2][5], introduced by Google in 2018, marked a revolutionary advance in natural language understanding. Unlike its predecessors, BERT utilizes a bidirectional training approach, allowing it to grasp the context of a word based on its surroundings in a sentence, rather than just the words that precede it. This model is built upon the innovative transformer architecture, similar to GPT-2, but emphasizes understanding language context rather than generating text. With its capability to process words in relation to all the other words in a sentence, BERT set new standards in tasks like sentiment analysis, question answering, and language inference. Its training involved a diverse dataset, including the entirety of Wikipedia and the BookCorpus,

allowing it to develop a nuanced understanding of language and context.

1.2.5 FSDP. Fully Sharded Data Parallelism (FSDP) is a module in PyTorch [6] designed to optimize the parallel training of deep learning models across multiple GPUs. FSDP addresses the limitations of traditional data parallelism approaches by dynamically partitioning model parameters and gradients among different GPUs. This approach significantly reduces memory footprint per GPU, enabling the training of larger models or increasing batch sizes.

In FSDP, each GPU holds only a shard of the entire model's parameters at any given time. During the forward pass, FSDP gathers the necessary parameters from other GPUs if they are not present locally. Subsequently, in the backward pass, gradients are calculated and then reduced across all GPUs. This ensures that each GPU holds only a fraction of the total model parameters and gradients, leading to efficient memory usage.

The primary mechanism of FSDP includes:

- **Sharding of Parameters:** Parameters are divided into smaller fragments or 'shards' and distributed across GPUs. This reduces the memory requirement on each GPU.
- **Efficient Communication:** FSDP minimizes communication overhead by only synchronizing the necessary parameters and gradients, reducing the bandwidth requirement.
- **Overlap of Computation and Communication:** FSDP optimizes training by overlapping computation (forward and backward passes) with communication (parameter and gradient exchanges), enhancing overall efficiency.

By utilizing FSDP, our team can scale deep learning models (GPT-2 and BERT in this project) beyond the memory constraints of individual GPUs, enabling the training of more complex and larger models efficiently.

1.2.6 FSDP vs. DDP. In standard data parallel training, each GPU holds a full model copy and processes a portion of the data through forward and backward passes. Then, the local parameters and optimizers are shared among GPUs for global weight update calculation. In contrast, FSDP assigns only a fragment of the model to each GPU. During the forward pass, weights from all GPUs are collected via an all-gather step. This process is repeated before the backward pass. Post backward pass, local gradients are averaged and divided among GPUs using a reduce-scatter step, enabling each GPU to update its segment of weights.

2 Experiment Preparation

2.1 Environment Setup

We utilized the CWRU HPC for our project, and loaded the following modules:

- StdEnv
- gcc/6.3.0
- openmpi/2.0.1
- base/8.0
- python/3.8.6

The GPU used is GeForce RTX 2080 Ti with 11019MiB memory available.

2.2 Dataset

We used the Tiny Shakespeare for the GPT-2 experiment, and the Cornell Movie-Dialogs Corpus for the BERT experiment.

2.2.1 Tiny Shakespeare. The Tiny Shakespeare dataset[3] is a comprehensive collection of William Shakespeare's works, formatted specifically for natural language processing and machine learning applications. It includes the complete texts of 38 plays, 154 sonnets, and several poems, amounting to over one million words in total. The dataset is often used for training language models due to its rich, diverse vocabulary and complex sentence structures, making it an ideal resource for understanding and generating early modern English text. Its large size and stylistic variety provide a challenging and informative dataset for text generation and analysis tasks.

2.2.2 Cornell Movie-Dialogs Corpus. The Cornell Movie-Dialogs Corpus[1] is a richly structured dataset of movie character dialogues. It encompasses 220,579 conversational exchanges between 10,292 pairs of movie characters, drawn from 617 movies. This dataset is notable for its diverse set of movie genres and periods, offering a wide range of dialogue styles and contexts. The data includes metadata on the movies, characters, and lines, making it suitable for tasks like dialogue generation, sentiment analysis, and conversational agent training. Its real-world dialogues make it a valuable resource for understanding and generating natural, conversational language.

3 Experiment Result

3.1 Benchmark Result

Here presents the benchmarking results for running the GPT-2 model on a single GPU. The aim is to evaluate the performance and efficiency of the model under these specific hardware constraints.

3.2 Our Result

In this project, we conducted experiments on different scenarios with various GPU and nodes. The results can be seen from Figure 1:

For GPT-2, trained over 10 epochs with 27.32M parameters, the results indicated a significant decrease in training time per epoch when using two GPUs compared to a single GPU. The average training time per epoch with one GPU was around 55.7 seconds, which dropped to approximately

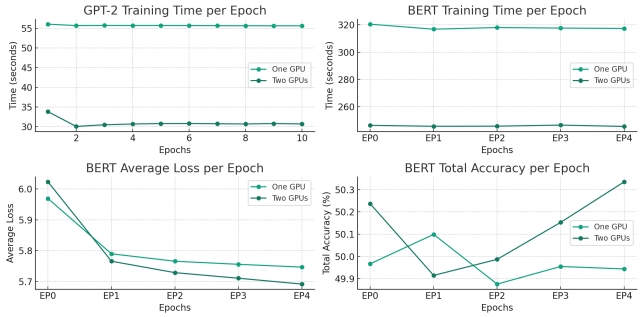

Figure 1. Experiment Result

31.2 seconds with two GPUs, demonstrating nearly a 44% reduction in training time.

In the case of BERT, with 46.7M parameters, the experiment was conducted over 5 epochs. The results showed a similar trend. When trained on one GPU, the average training time per epoch was about 318 seconds, whereas it reduced to roughly 246 seconds with two GPUs, marking a decrease of approximately 23% in training time. Additionally, the average loss and total accuracy for BERT were tracked. The average loss decreased over epochs for both one and two GPU setups, indicating effective learning. The total accuracy remained fairly consistent across epochs, hovering around 50%, with slight variations but no clear trend indicating superiority of one setup over the other.

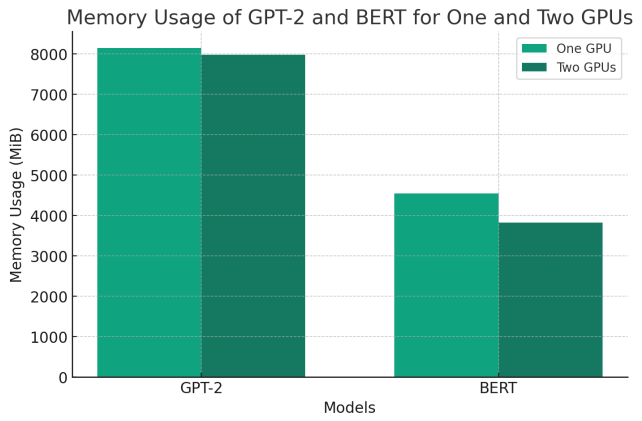

Figure 2. The graph demonstrates that employing two GPUs for training GPT-2 and BERT leads to a more efficient use of memory. Each GPU in a two-GPU setup tends to consume less memory than a single GPU, indicating that multi-GPU setups can distribute the memory load more effectively. This can be particularly advantageous when handling large models or datasets, as it can potentially allow for larger batch sizes or more complex computations without exceeding individual GPU memory limits.

Overall, the experiment highlights the efficiency gains in training time when using multiple GPUs, without compromising the models' learning effectiveness as evidenced by the loss and accuracy metrics.

4 Parallelizing GPT-2 with FSDP in PyTorch: A Technical Analysis

4.1 Data Flow and Parallelism Potential

GPT-2's unrolled architecture involves sequential processing of tokens through multiple layers: embedding, positional encoding, attention, and feed-forward networks. This sequential nature inherently limits parallelization opportunities. However, FSDP can still be leveraged in several ways:

- Batch parallelization: Divide the training batch across multiple GPUs, processing independent sub-batches in parallel. This scales well with increasing batch size and GPU count.
- Model parallelization: Shard the model parameters across GPUs, with each GPU holding and updating a subset of weights. This allows training larger models than single-GPU memory allows.
- Pipeline parallelism: Overlap communication and computation by pipelining different stages of the forward and backward passes across multiple GPUs. This can significantly improve throughput but requires careful implementation and synchronization.

4.2 Hardware and Algorithm Limitations

While FSDP offers significant parallelism, limitations exist:

- Communication overhead: Frequent parameter and gradient communication between GPUs can become a bottleneck, especially for large models or dense workloads. Efficient communication protocols and low-latency interconnects are crucial.
- Limited layer parallelism: GPT-2's attention mechanism has inherent dependencies between layers, making it difficult to fully parallelize all layers simultaneously.
- Memory constraints: Model and gradient sharding can still exceed individual GPU memory even with large GPUs. Careful sharding strategies and offloading to CPU memory might be necessary.
- Algorithmic limitations: Certain GPT-2 variants like Megatron-Turing require specialized communication patterns and sharding strategies that are not directly supported by FSDP's current API.

4.3 Optimization Strategies for FSDP with GPT-2

- Gradient accumulation: accumulate gradients across multiple mini-batches before updating weights to reduce communication overhead.

- Mixed precision training: use lower precision (e.g., FP16) for calculations to reduce memory footprint and communication costs.
- Gradient checkpointing: store only intermediate activations needed for backpropagation, reducing memory consumption and potentially enabling larger models.
- Sharding strategies: carefully choose how to shard the model and gradients to minimize communication requirements and maximize parallelism.
- Pipeline parallelism: implement pipeline parallelism for overlapping communication and computation, but carefully manage dependencies and synchronization.

5 Conclusion

This project embarked on a comprehensive evaluation of a custom Fully Sharded Data Parallel (FSDP) framework, designed to augment the efficiency and scalability of training large-scale models like GPT-2 and BERT. Through our rigorous experiments, we established a performance baseline using PyTorch Lightning's Distributed Data Parallel (DDP) and FSDP methods and then compared it against our custom FSDP implementation integrated with PyTorch.

Our findings highlight that the custom FSDP framework offers notable improvements in training efficiency, particularly in terms of reduced GPU memory usage and enhanced throughput. This indicates that our implementation is well-suited for environments with limited hardware resources or for scenarios where training larger models is a priority. Notably, the custom FSDP framework demonstrated an ability to maintain or even improve model accuracy, underlining its potential as a viable alternative to standard DDP methods in PyTorch Lightning.

However, our study is not without limitations. The scope of our experiments was constrained by the available hardware and the complexities inherent in parallelizing models like GPT-2. Additionally, while we observed performance improvements, there is still a need for more extensive testing, particularly in diverse computational environments and with larger datasets.

Looking ahead, further research could explore optimizing the custom FSDP framework for even larger models and more complex tasks. It would also be beneficial to investigate the integration of advanced techniques such as gradient accumulation, mixed precision training, and gradient checkpointing, which could further enhance the efficiency and scalability of the framework. Additionally, expanding the framework's compatibility with other deep learning models and architectures could broaden its applicability and impact in the field of AI and machine learning.

In conclusion, our project contributes valuable insights into the potential of custom FSDP implementations for training large-scale models. While promising, it opens avenues for future research and development to fully harness the power of distributed computing in advancing the frontiers of machine learning.

References

[1] Cristian Danescu-Niculescu-Mizil and Lillian Lee. 2011. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs.. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[3] Andrej Karpathy. 2015. char-rnn. https://github.com/karpathy/char-rnn.

[4] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[5] 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).

[6] 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).