Profiling Parallelism: A Tool for Efficient Model Deployment on GPUs

Peikun Guo
pg34@rice.edu

Jie-Si Chen
jc182@rice.edu

I. Introduction

As the demand and reliance on larger machine learning models grow, especially foundational models such as GPT [1], Llama [2], and diffusion models [3], there arises a challenge: How to effectively deploy these models on varied machine setups, especially GPU setups? With the plethora of deployment strategies and paradigms of parallelism available, users are often left unsure of which technique optimally suits their specific environment, hindering efficiency and performance. Current methodologies, such as data parallelism, model parallelism [4], tensor parallelism [5], and pipeline parallelism, are well-researched and offer approaches for maximizing GPU utilization. However, the missing piece is a tailored guide on a given user's end that evaluates the performance of these techniques for a specific combination of model and hardware. Presently, users must undertake a trial-and-error process, consuming valuable time and resources, to determine the best approach for their unique setup.

Our goal is to develop a dynamic profiling utility, which works like a probe before full-scale model deployment, and runs proxy tasks to evaluate different parallel deployment strategies. For example, in a inference-only use case, this utility will perform iterations of forward propagation across varied parallelism techniques, providing users with a comprehensive performance profile for their specific model and hardware configuration. We propose to design and implement a fast profiling tool that assesses the efficiency of different parallelism strategies tailored to a user's specific environment. Specifically, by performing proxy tasks, our tool will generate a performance profile, highlighting both computational operations (matrix multiplication, convolution) and GPU communication (ops such as gather, reduce) runtimes. This will enable users to make informed decisions on the optimal deployment strategy for their machine learning models without undergoing a tedious trial-and-error process.

Our evaluation will be two-fold: first, the efficacy of the profiling tool itself and second, its applicability across varied environments. We will be able deploy our tool on two machines: one with two NVIDIA 3090 GPUs and another with two NVIDIA 2080ti GPUs. By leveraging various mainstream models of different sizes, we aim to demonstrate the effect of environment heterogeneity on the choice of parallelism paradigms for distributed model training and inference. The results will provide insight into how different hardware setups and model complexities interact with varied parallelism techniques.

Our work intends to streamline the process of deploying large machine learning models by offering a quick and insightful profiling tool. This will greatly reduce the decision-making time for users, enabling more efficient utilization of GPU resources. Additionally, by understanding the nuances of different deployment strategies in varied environments, our findings will contribute to the broader discussion on optimized model deployment.

II. Background

A. Paradigms of Parallelism in Deep Models

The continual expansion of deep learning models has brought about an increasing urgency for parallel training. Single-GPU training is no longer sufficient given the parameter numbers of current models often exceed billion or even 10B. Parallel training strategies ensure that training time is manageable and efficient. We will introduce three mainstream parallelism paradigms: data parallelism, tensor parallelism and pipeline parallelism. Other techniques such as optimizer-level parallelism and parallelism on heterogeneous systems will not be the focus of this project. The figures in this section are from the document of the open source project ColossalAI [6].

1) Data Parallelism: In common implementations of data parallelism such as [7], the dataset is divided into multiple shards, with each shard being assigned to a distinct device. Each device maintains a complete replica of the model and processes its specific shard of the dataset. After back-propagation, the model gradients undergo an "all-reduce" operation, ensuring synchronization of model parameters across all devices.

2) Tensor Parallelism: Data parallelism leads to redundancy since every GPU possesses the entire model's weights [8]. To circumvent this redundancy, model parallelism was developed, in which the model is partitioned and distributed over multiple devices. First subtype of model parallelism is tensor parallelism. It divides a tensor into chunks along a certain dimension. Each device retains only a fraction of the whole tensor without compromising the computational graph's integrity. A common example is matrix multiplication. The matrix is split and distributed across devices, ensuring that computations are correctly executed across all devices [8], [9].

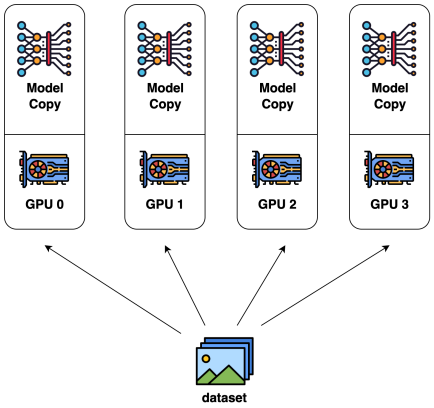

Fig. 1. Data parallelism

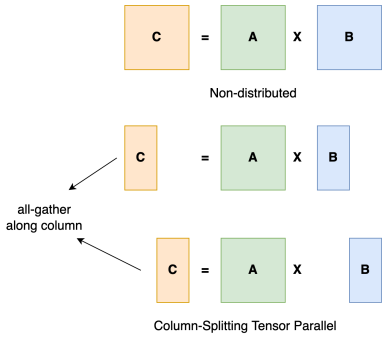

Fig. 2. Data parallelism

3) Pipeline Parallelism: Another subtype of model parallelism is pipeline parallelism. Drawing inspiration from CPU architecture, this approach partitions the model by layers [10], [11]. Each layer or set of layers is allocated to a specific device. During the forward propagation, each device transfers its intermediate activations to the next stage. For backward propagation, gradients of the input tensor are passed back to the preceding stage. This parallel processing boosts training throughput. One of the challenges is the "bubble time" - intervals when some devices are idly waiting, leading to potential inefficiencies [9], [12].

B. Existing Framework

Parallel training and inference has seen significant advancements in recent times with several notable frameworks emerging to help researchers and engineers tackle the challenges of training large-scale models. The complexity of implementing them is very high, and from the standpoint of not rebuilding the wheels, we will discuss three of the most prominent frameworks, namely NVIDIA's Megatron-LM [9], Meta's DeepSpeed [13], and ColossalAI [6].

Megatron-LM [9] is NVIDIA's brainchild and is tailored for training massive transformer models. This framework shines in its ability to handle gigantic language models by leveraging

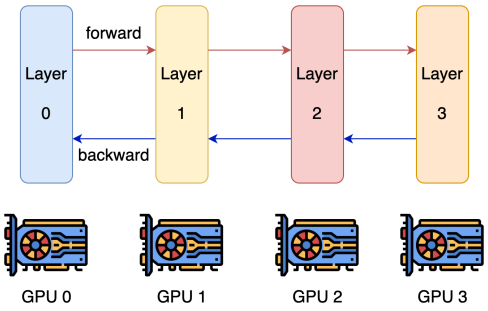

Fig. 3. Data parallelism

model parallelism. It primarily focuses on data parallelism, pipeline parallelism, and 1D tensor parallelism. DeepSpeed [13] is another notable player in the domain of distributed deep learning. This open-source deep learning optimization library is designed to make distributed training easy while reducing memory usage and improving the scale and speed of models. It comes with several innovations such as ZeRO (Zero Redundancy Optimizer) [14] to optimize memory usage and achieve super-linear scaling of training time across machines.

ColossalAI [6] is a distributed training framework developed by the HPC-AI tech team. Unlike Megatron-LM and DeepSpeed, which primarily focus on data parallelism, pipeline parallelism, and 1D tensor parallelism, ColossalAI integrates all these and additionally offers infinite-dimensional tensor parallelism. Moreover, in the context of this course project, ColossalAI offers an easy-to-use API and higher customizability than Megatron-LM or DeepSpeed for we developers to adapt the system to different parallelism strategies.

III. Method and Evaluation Plans

A. Framework

The backbone of our proposed system is the dynamic profiling utility. It functions akin to a probe, gauging the environment and model before full-scale deployment. We have designed proxy tasks (e.g. forward and back propagation on synthetic data) to replicate typical model deployment operations without running the model in its entirety. These tasks provide a sampling of what the full-scale operations would entail, allowing us and potential users to gather performance metrics with minimal resource consumption.

B. Profiling Workflow

Given the unique challenges posed by varied machine learning models and hardware configurations, our utility will execute iterations of forward propagation (for inference-only use cases) across multiple parallelism strategies. Each iteration will document the run time of computational operations, primarily including matrix multiplication and convolution, as well as GPU communication tasks like gather and reduce operations. We will carefully choose the specific tool (e.g.

hooks in the functions; exisiting profiling tool) to time the executions.

C. Expected Resources

Our primary aim is to validate the reliability and efficacy of our profiling utility on mainstream CUDA-supported GPUs. To do so, we will deploy the tool on two hardware configurations: a machine equipped with two NVIDIA 3090 GPUs and another with two NVIDIA 2080ti GPUs. To ensure a comprehensive evaluation, we will deploy various mainstream models ranging in complexity and size.

D. Data (Model) Selection

In the context of our project, the data, or the "raw materials" to be profiled, are in fact the models. In our case, we do not care for models' ability or performance, only their inference speed or back propagation speed under distributed multi-GPU setup. To ensure a comprehensive evaluation, we will deploy various mainstream models ranging in complexity and size. This will include, but not be limited to, foundational models such as GPT-3 [1], Llama [2], smaller-scale models like BERT [15], and diffusion models [3].

E. Performance Metrics

The utility's effectiveness will be measured by comparing the predictions of our profiling tool against (1) actual deployment runtimes and performance metrics and (2) existing empirical studies or evaluations that compared the parallelism paradigms. Key performance indicators will include accuracy of profiler-predicted GPU utilization, computational operation times, GPU communication overhead, and overall deployment efficiency.

IV. Timeline / Milestones

The project spans over a six-week duration. Below is a tentative breakdown (as of the week of Oct 9):

- **Week 1: Setup and Initial Design** Set up the environments on the machines; finalize machine learning models for evaluation and download their weights. Sketch the primary design of the dynamic profiling utility.
- **Week 2: Development of Core Profiling Features** Implement computational operation profiling and design proxy tasks for forward propagation evaluations.
- **Week 3: Feedback Taking and Refinement** Conduct initial tests, discuss within the team and with the processor, gather feedback, and refine the profiling utility and the whole workflow accordingly.
- **Week 4: Experimenting with Profiling Utility** Deploy the utility on testbeds (NVIDIA 3090 and 2080ti GPUs), gather results, and identify areas for improvement.
- **Week 5: Comprehensive Evaluation** Conduct extensive tests across varied models and hardware setups, then collate and refine results.
- **Week 6: Report Writing** Compile findings into a report and finalize it for submission.

References

[1] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[4] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks." *Proceedings of Machine Learning and Systems*, vol. 1, pp. 1–13, 2019.

[5] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Hypar: Towards hybrid parallelism for deep learning accelerator array," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 56–68.

[6] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-ai: A unified deep learning system for large-scale parallel training," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 766–775.

[7] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[8] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[9] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[10] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[11] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[12] S. Li and T. Hoefler, "Chimera: efficiently training large-scale neural networks with bidirectional pipelines," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.

[13] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. 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*, 2020, pp. 3505–3506.

[14] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.