POSTER: Pattern-Aware Sparse Communication for Scalable Recommendation Model Training

Jiaao He
Tsinghua University
Beijing, China
hja20@mails.tsinghua.edu.cn

Shengqi Chen
Tsinghua University
Beijing, China
csq20@mails.tsinghua.edu.cn

Jidong Zhai
Tsinghua University
Beijing, China
zhaijidong@tsinghua.edu.cn

Abstract

Recommendation models are an important category of deep learning models whose size is growing enormous. They consist of a sparse part with TBs of memory footprint and a dense part that demands PFLOPs of computing capability to train. Unfortunately, the high sparse communication cost to re-organize data for different parallel strategies of the two parts impedes the scalability in training.

Based on observations of sparse access patterns, we design a two-fold fine-grained parallel strategy to accelerate sparse communication. A performance model is built to select an optimal set of items that are replicated across all GPUs so that all-to-all communication volume is reduced, while keeping memory consumption acceptable. The all-to-all overhead is further reduced by parallel scheduling techniques. In our evaluation on 32 GPUs over real-world datasets, $2.16-16.8\times$ end-to-end speedup is achieved over the baselines.

CCS Concepts: • **Computing methodologies** → **Massively parallel algorithms**; • **Computer systems organization** → *Neural networks.*

Keywords: Distributed Deep Learning, Parallelism

1 Introduction

Making more accurate recommendations has always been one of the most important objectives in the industry. Recommendation models based on deep learning (DLRM) are showing superior model quality. They automatically learn from the enormous data generated by users' online behaviors. For instance, websites predict the click-through rate (CTR) of a video to decide whether to show it to a user. And then, actual click data are used to train CTR-predicting models. This training cycle is processed every few minutes for timely and precise recommendations.

PPoPP '24, March 2–6, 2024, Edinburgh, United Kingdom

ACM ISBN 979-8-4007-0435-2/24/03.
https://doi.org/10.1145/3627535.3638481

As shown in Figure 1, a DLRM model mainly consists of two parts. (a) *Sparse Part* contains one or multiple embedding tables. They map discrete IDs of users, videos, or any other modeled items to embedding vectors, which commonly consist of tens of floating point numbers. (b) *Dense Part* consists of various typical neural network models, which are applied to the embedding vectors of input items, explore the relationship among them, and make predictions.

Scaling up DLRM training is important but challenging. In production, DLRMs contain up to trillions of parameters [6, 7], requiring hundreds of GPUs to train. In fact, two different aspects of scalability are required simultaneously, related to two parts of DLRMs. (a) The sparsely-accessed embedding tables contain terabytes of embedding vectors, so they require extensive memory capacity. (b) The dense part is compute-intensive and appreciates scalable throughput.

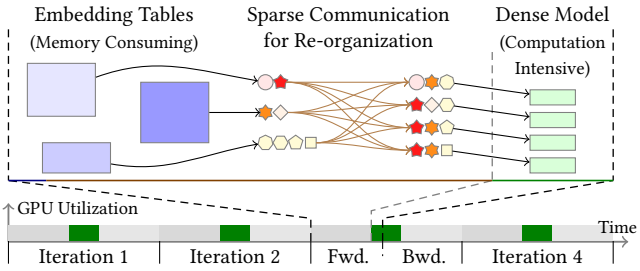

Figure 1. Sketch of a DLRM Training Process

To meet different scalability requirements across the two parts of DLRM, two different parallel strategies are typically applied: (a) The memory-consuming embedding tables are partitioned across workers. (b) The dense models with intensive computation are replicated.

In training, however, every sample is related to tens of embedding vectors that may reside in many other workers. As revealed in Figure 1, the sparse communication of re-organizing embedding vectors (from the place where they are stored to GPUs where they are used for training) becomes a performance bottleneck.

2 Observation: Skewed Sparse Access

Intuitively, accessing to the items in DLRM training data is skewed. Some items are much more frequently accessed than others, e.g., accounts of celebrities or spot news.

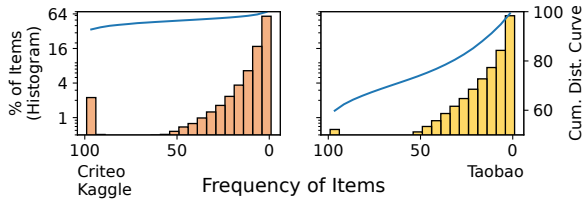

Figure 2. Observation on Skewed Distribution of Items

Two public datasets, *Criteo* [3] and *Taobao* [1], are inspected by sampling 100 batches from each of them. The histogram in Figure 2 aggregates the frequency of items, i.e. the number of batches containing the item out of the 100 batches. In *Criteo* dataset, 2.2% items appear in more than 95 batches, indicated by the left-most bump.

We further inspect another batch to obtain the distribution of items over each bin, which indicates embedding vector access during training. Fewer than 1% items are not present in the previous 100 batches. As the cumulative curves in Figure 2 suggest, in *Criteo Kaggle* dataset, more than 90% accesses belong to the 2.2% frequent items. While in *Taobao*, 13% items can cover about 86% embedding vector accesses. Once these items are replicated on every GPU, the all-to-all communication volume can be significantly reduced.

3 Overview of P²Res

In our system, P²Res, we decompose the sparse communication into two parts following an item-wise two-fold parallel strategy (*2FP*). Figure 3 illustrates the design of *2FP*. There are two exclusive types of partitions. Any embedding vector belongs to exactly one of them: $\mathcal{R}$, in which data parallelism is used on a small portion of frequent items; or $\mathcal{H}_i$, in which model parallelism handles the rest items.

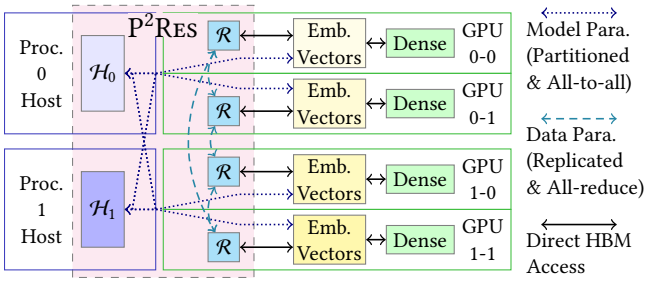

Figure 3. Placement of the Two Partitions in *2FP*

$\mathcal{R}$ is the replicated partition for high throughput. There is a replica of $\mathcal{R}$ on each GPU, so accesses of embedding vectors in $\mathcal{R}$ are performed locally through the high-bandwidth device memory (HBM). Inter-device data movement is eliminated during forward and backward propagation, removing significant communication overhead. All-reduce is used by $\mathcal{R}$ to synchronously train the model. A performance model is applied to choose the most appropriate items for $\mathcal{R}$, minimizing overall communication volume considering both model setup and hardware environment.

$\mathcal{H}_i$ $(0 \le i < N_{\text{proc.}})$ denotes the high-capacity partition on process i. Host memory is used by the process to store embedding vectors in $\mathcal{H}_i$. This saves GPU memory to store $\mathcal{R}$ and the dense part. During training, processes fetch embedding vectors of samples in its local batch from all $\mathcal{H}_i$, excluding those in $\mathcal{R}$. To make these sparse operations more efficient, we optimize the distributed schedule of embedding vector indexing and data exchanging.

4 Evaluation Results

We implement P²Res using MPI and CUDA. It is evaluated on 32 NVIDIA V100 PCIe GPUs connected by Infiniband.

We compare the performance of P²Res with three open-source baselines. **TFDE** [10] implements Parallax [5] that maintains the sparse part by a distributed sparse parameter server. **TorchRec** [4] uses a coarse-grained hybrid parallel strategy [7, 9] to utilize the skewness across embedding tables. **HugeCTR** [11] places all embedding tables on GPUs, and utilizes GPU-direct communication techniques to accelerate all-to-all communication.

The test cases include three public real-world datasets for three popular DLRMs, as listed in Table 1.

Table 1. Datasets and Models for Evaluation

Dataset	Taobao[1]	Criteo Kaggle[3]	Terabytes [2]
Model	DCN[8]	Legacy[11]	DLRM[12]
# Feat. (Den./Sp.)	4/4	1/39	14/26
Embedding Size	117MB	411MB	96.1GB

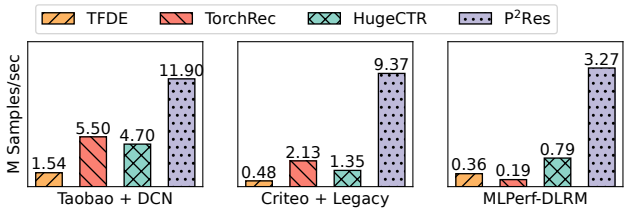

Figure 4. End-to-end Training Throughput on 32 GPUs

As shown in Figure 4, P²Res outperforms all baselines on all test cases, achieving up to 16.8× speed up.

5 Conclusion

This paper introduces a two-fold item-wise hybrid parallel strategy for DLRM training. A performance model selects certain items to replicate across all GPUs for minimum communication latency. The rest sparse access across processes is optimized by better scheduling tasks on the host. P²Res is transparent to the model, and achieves competitive scalability over the baselines.

Acknowledgement

This work is supported by National Key R&D Program of China under Grant 2021ZD0110104 and National Natural Science Foundation of China (62225206, U20A20226). Jidong Zhai is the corresponding author.

References

[1] Alimama. 2020. Ad Display/Click Data on Taobao.com. https://www.kaggle.com/datasets/pavansanagapati/ad-displayclick-data-on-taobaocom.

[2] Criteo. 2014. Criteo 1TB Click Logs Dataset. https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset.

[3] Criteo. 2014. Criteo Display Advertising Challenge. http://www.kaggle.com/c/criteo-display-ad-challenge.

[4] Dmytro Ivchenko, Dennis Van Der Staay, Colin Taylor, Xing Liu, Will Feng, Rahul Kindi, Anirudh Sudarshan, and Shahin Sefati. 2022. TorchRec: a PyTorch Domain Library for Recommendation Systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 482–483.

[5] Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, and Byung-Gon Chun. 2019. Parallax: Sparsity-aware data parallel training of deep neural networks. In *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*, George Candea, Robbert van Renesse, and Christof Fetzer (Eds.). ACM, 43:1–43:15. https://doi.org/10.1145/3302424.3303957

[6] Xiangru Lian, Binhang Yuan, Xuefeng Zhu, Yulong Wang, Yongjun He, Honghuan Wu, Lei Sun, Haodong Lyu, Chengjun Liu, Xing Dong, et al. 2022. Persia: An open, hybrid system scaling deep learning-based recommenders up to 100 trillion parameters. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3288–3298.

[7] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 993–1011.

[8] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[9] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. 2022. RecShard: statistical feature-based memory optimization for industry-scale neural recommendation. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 344–358.

[10] Shashank Verma, Wenwen Gao, Hao Wu, Deyu Fu, and Tomasz Grel. 2022. Fast, Terabyte-Scale Recommender Training Made Easy with NVIDIA Merlin Distributed-Embeddings. https://github.com/NVIDIA-Merlin/distributed-embeddings.

[11] Zehuan Wang, Yingcan Wei, Minseok Lee, Matthias Langer, Fan Yu, Jie Liu, Shijie Liu, Daniel G Abel, Xu Guo, Jianbing Dong, et al. 2022. Merlin HugeCTR: GPU-accelerated Recommender System Training and Inference. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 534–537.

[12] Carole-Jean Wu, Robin Burke, Ed H Chi, Joseph Konstan, Julian McAuley, Yves Raimond, and Hao Zhang. 2020. Developing a recommendation benchmark for MLPerf training and inference. *arXiv preprint arXiv:2003.07336* (2020).