Scalable GNN Training for Track Finding

Paolo Calafiura[2], Chen-Hsun Chan[2], Xiangyang Ju[2], Ivan A Ladutska[1], Alina Lazar[1], Daniel Murnane[4], Tuan Minh Pham[3], Brenden W Reeves[1]

[1]Youngstown State University, [2]Lawrence Berkeley National Lab, [3]University of Wisconsin-Madison, Madison, USA, [4]University of Copenhagen, Copenhagen, Denmark,

E-mail: alazar@ysu.edu

Abstract. Graph Neural Networks (GNNs) have demonstrated significant performance in addressing the particle track-finding problem in High-Energy Physics (HEP). Traditional algorithms exhibit high computational complexity in this domain as the number of particles increases. This paper addresses the challenges of training GNN models on large and rapidly evolving datasets, a common scenario given the advances in data generation, collection, and increasing storage capabilities. The computational and GPU memory requirements present significant roadblocks in efficiently training GNNs on large graph structures. One effective strategy to reduce training time is distributed data parallelism on multi-GPUs, which involves averaging gradients across the devices used for training. This paper presents the results of GNN training when using distributed data parallelism with an increased number of GPUs. Training scalability is analyzed via training time, speed_up, parallel efficiency, GPU and memory utilization. Using weak scaling for GNN training with distributed data parallelism leads to an increase in the physics performance. We are investigating the relationship between the number of devices and the accuracy of the resulting models. Preliminary results on the TrackML dataset are reported. GPU nodes from Perlmutter at NERSC are used to run the experiments.

1. Introduction

While deep learning offers numerous advantages and has achieved remarkable success in many domains, including High Energy Physics (HEP) [1], it is essential to note that it does not encompass a simple solution to all problems. Some challenges, such as interpretability, overfitting, and the large amounts of labeled data required, still exist. As HEP experiments continue to evolve, the generation and collection of data have surged, leading to increasingly large datasets that demand sophisticated processing techniques. This paper addresses the challenges associated with training GNN models [2] on such extensive and dynamically growing datasets, particularly highlighting the computational and GPU memory constraints that hinder efficient training on large graph structures.

Training performant GNN models for track reconstruction [3–5] may take days to weeks to complete. Speeding up this training is crucial for iterative development, experimentation, and finding the best hyperparameters. Hardware accelerators such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Intelligence Processing Units (IPUs) are specifically designed for the massive parallelism required by deep learning operations. Training using these hardware accelerators can be significantly faster than using CPUs. Distributing the training process across multiple GPUs and even multiple computing nodes provides a way to

train large models faster for both machine [6] and deep learning [7]. Frameworks like TensorFlow and PyTorch support such distributed training [8], allowing for the exploitation of large-scale computing systems, which is crucial for the iterative development and experimentation needed to optimize hyperparameters and achieve optimal model performance.

2. Distributed Data Parallelism (DDP)

Distributed Data Parallel (DDP) [9] deep learning training is the process used to train deep learning models on multiple devices or nodes. As datasets grow, it becomes increasingly challenging to fit them into the memory of a single machine [10]. This problem is further intensified when training GNN models for track reconstruction, primarily due to the irregular sizes of input graph samples. Distributing the dataset across multiple devices allows for parallel processing and, at the same time, allows for handling much larger datasets.

The training time can be significantly reduced by distributing the training process across multiple GPUs or devices. For example, one of the most popular foundation models, the OpenAI's GPT-4 model [11], was trained for 100 days on around 25,000 Nvidia A100 GPUs. In the case of the DDP approach, each device receives a subset of the dataset, computes the gradients, as shown in Figure 1, and then the gradients are aggregated and averaged to update the global model at each iteration.

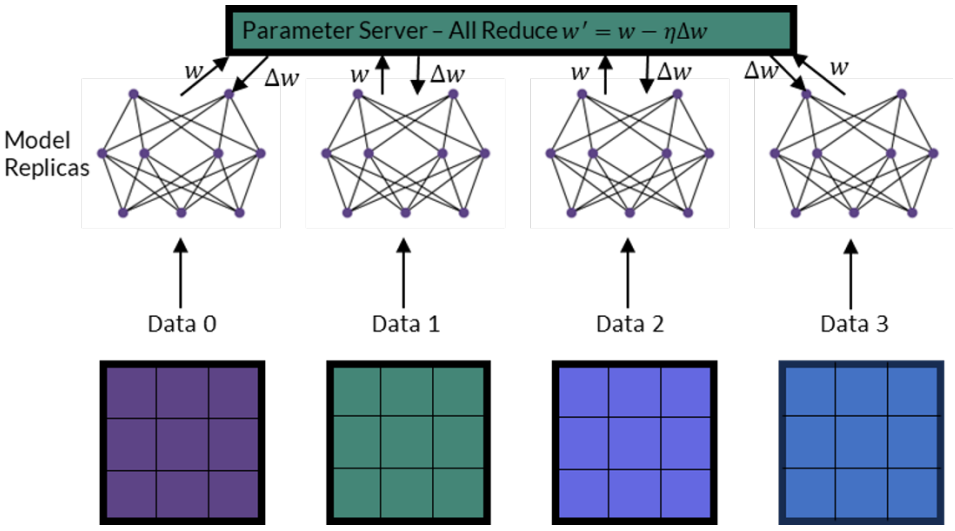

Figure 1. Each process performs a full backward pass in parallel.

This DDP approach allows for efficient parallelization of data across multiple GPUs, facilitating faster training times and improved resource utilization. The choice of DDP was driven by its robust performance in maintaining model accuracy and its ability to handle large-scale datasets effectively. With the help of the PyTorch Lightning framework, implementing DDP has enabled us to distribute the training workload evenly across available GPUs, thereby accelerating the training process and ensuring consistency in model updates.

In general, scaling methods can be divided into two types: weak and strong scaling. Weak scaling involves increasing the training dataset size proportionally with the number of GPUs, which theoretically should maintain constant training time. We also observe that as the number of GPUs increases, the convergence of training increases due to the corresponding increase in the global batch size. Scaling the learning rate with the batch size is recommended in order to help maintain convergence rates and ensure that the model continues to learn effectively despite the larger batch sizes.

By distributing the model and data across multiple devices, you can train models that would not fit into the memory of a single GPU. Some research suggests that training models in a distributed manner, especially with asynchronous updates, can introduce a form of implicit regularization, potentially leading to better generalization of the test dataset.

As more computational power is needed, more nodes can be added to the training cluster. Cloud providers offer a variety of GPU instances. Distributed training allows for flexibility in choosing a combination of instances that might be more cost-effective than using a single powerful and more expensive instance. Distributed training can run multiple experiments in parallel, allowing researchers to explore more hyperparameter settings or model architectures in a shorter time.

Distributed data parallel deep learning training is essential for handling modern deep learning tasks' increasing complexity and size. It offers a way to efficiently utilize resources, speed up training, and handle more extensive datasets and models.

The main steps to run DDP are:

(i) Each GPU across each node gets its own process.

(ii) Each GPU gets visibility into a subset of the overall dataset. It will only ever see and train on that subset.

(iii) Each process initializes a copy of the model.

(iv) Each process performs a full forward and backward pass in parallel as shown in Figure 1.

(v) The gradients are synchronized and averaged across all processes by performing a collective all-reduce operation.

3. Dataset

This work uses the TrackML dataset [12] for development and performance evaluation. The open-access TrackML dataset, containing simulated events from a proton collider tracking detector, was designed to develop algorithms to reconstruct particle trajectories in HEP experiments. This dataset enables researchers to utilize realistic datasets to build their models, enhancing the relevance and applicability of their findings to practical scientific challenges.

- The **TrackML** dataset consists of independent events generated through a Monte Carlo simulation of proton-proton collisions.
- The full dataset contains **10,000** events. During pre-processing, we convert these coordinates to spherical coordinates.
- This dataset is notable because the simulations give us information about the **ground truth**: we know the "actual" tracks of each particle. We can compute the physics performance in terms of efficiency and purity.

4. Distributed Data Parallelism (DDP) Experiments and Results

A weak-type scaling experiment has been designed to investigate the scalability of the GNN model training. Weak scaling means that the problem size increases as the number of devices is increased. The baseline problem size for training on a single GPU is 100 events for training and 20 events for validation. When the number of devices is increased, the size of the training and validation sets is increased accordingly, as shown in Table 2. The training was run for 50 epochs for each experiment. Given the memory requirements for the graphs that represent the events, we use a batch size of one for training. The results in Table 1 show how each physics performance measure (efficiency, purity, and area under the curve (AUC)) increases with the number of GPUs. The time per epoch and per batch also increases.

DDP can significantly reduce the training time by distributing the workload across multiple devices. However, as shown in Figure 2 (a), the speed-up is not linear with the number of

GPUs. For 32 GPUs, the speed-up is approximately 22x. GPU utilization increases from 60% for one GPU to 81% when training is run on 32 GPUs while memory utilization varies less.

Using DDP effectively, you can process more data and increase performance, as shown in Figure 2 (b) and Table 1. Efficiency increases as the global batch size increases. Purity and AUC also increase. However, we observe a decrease of these measures going from 16 GPUs to 32 GPUs.

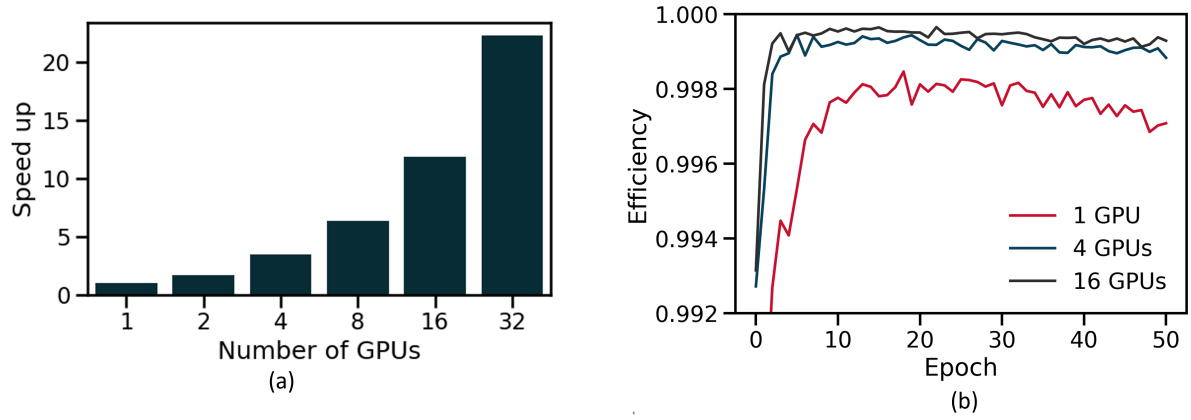

Figure 2. Speed up of running DDP (a) and efficiency on the validation dataset (b)

Table 1. Validation Performance Metrics during Training and Runtime

# of GPUs	Efficiency (%)	Target Purity	Total Purity (%)	Purity (%)	AUC	Validation Loss	Epoch (s)	Batch (s)
1	99.6	87.831	99.249	99.152	98.577	0.00406	34.36	0.24
2	99.778	88.431	99.513	99.452	98.842	0.00226	39.63	0.29
4	99.838	88.508	99.587	99.536	98.846	0.00176	38.73	0.28
8	99.871	88.524	99.658	99.616	98.841	0.00141	42.12	0.31
16	99.903	88.499	99.674	99.633	98.884	0.00114	45.48	0.34
32	99.945	87.931	99.443	99.371	98.79	0.00124	47.6	0.37

Table 2. Max GPU Utilization and Memory on the Main Node

GPUs	Train Events	Val Events	GPU Util (%)	Memory (GB)	Speed up
1	100	20	59.93	20.52	1
2	200	40	65.8	22.23	1.67
4	400	80	65.88	23.58	3.52
8	800	160	68.02	26.51	6.31
16	1600	320	73.91	26.18	11.86
32	3200	640	80.91	23.37	22.26

All the experiments were performed using the Perlmutter Supercomputer hosted by the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. Each Perlmutter node has a single 64-core AMD EPYC CPU and four NVIDIA A100 GPU accelerators with 40 GB and 80 GB of memory. We used PyTorch 2.3, PyG 2.5, and PyTorch Lightning 2.4 for the software stack.

5. Conclusion

The increasing complexity and volume of data in modern computational tasks necessitate adopting parallel and distributed computing techniques for deep learning training. This paper provides a scalability analysis of the distributed data-parallel training of the track reconstruction GNN model.

- **Data Parallelism:** DDP allows you to split your dataset across multiple GPUs (Figure 1).
- **Reduced Training Time:** By distributing the workload across multiple devices, DDP can significantly reduce the training time. However, as shown in Figure 2 (a) the speed-up is not linear with the number of GPUs.
- **Increase Performance:** By utilizing DDP effectively, you can process more data and increase performance as shown in figure 2 (b) and in Table 1.
- In the future, we plan to investigate the relationship between the number of devices and the increase in model efficiency and how to tune the learning rate for the DDP.

Acknowledgments

This research was supported in part by the U.S. Department of Energy's Office of Science, Office of High Energy Physics, of the US Department of Energy under Contracts No. DE-AC02-05CH11231 (CompHEP Exa.TrkX) and No. DE-SC0024364 (YSU FAIR).

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 and the Ohio Supercomputer Center (OSC).

References

[1] Kar D 2024 *The European Physical Journal Special Topics* 1–16

[2] Shao Y, Li H, Gu X, Yin H, Li Y, Miao X, Zhang W, Cui B and Chen L 2024 *ACM Computing Surveys* **56** 1–39

[3] Ju X, Murnane D, Calafiura P, Choma N, Conlon S, Farrell S, Xu Y, Spiropulu M, Vlimant J R, Aurisano A *et al.* 2021 *The European Physical Journal C* **81** 1–14

[4] DeZoort G, Battaglia P W, Biscarat C and Vlimant J R 2023 *Nature Reviews Physics* **5** 281–303

[5] Caillou S, Calafiura P, Ju X, Murnane D, Pham T, Rougier C, Stark J and Vallier A 2024 Physics performance of the atlas gnn4itk track reconstruction chain *EPJ Web of Conferences* vol 295 (EDP Sciences) p 03030

[6] Lazar A, Sim A and Wu K 2020 Gpu-based classification for wireless intrusion detection *Proceedings of the 2021 on Systems and Network Telemetry and Analytics* pp 27–31

[7] Dehghani M and Yazdanparast Z 2023 *Journal of Big Data* **10** 158

[8] Li S, Zhao Y, Varma R, Salpekar O, Noordhuis P, Li T, Paszke A, Smith J, Vaughan B, Damania P *et al.* 2020 *arXiv preprint arXiv:2006.15704*

[9] Aach M, Inanc E, Sarma R, Riedel M and Lintermann A 2023 *Journal of Big Data* **10** 1–23

[10] Lee C, Hewes V, Cerati G, Wang K, Aurisano A, Agrawal A, Choudhary A and Liao W K 2024 *Frontiers in High Performance Computing* **2** 1458674

[11] Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman F L, Almeida D, Altenschmidt J, Altman S, Anadkat S *et al.* 2023 *arXiv preprint arXiv:2303.08774*

[12] Kiehn M, Amrouche S, Calafiura P, Estrade V, Farrell S, Germain C, Gligorov V, Golling T, Gray H, Guyon I *et al.* 2019 *EPJ Web of Conferences* **214** 06037