Optimal Distributed Training With Co-Adaptive Data Parallelism in Heterogeneous Environments

Lifang Chen , **Zhichao Chen** , **Liqi Yan**[*] , **Yanyu Cheng** ,
Fangli Guan and **Pan Li**[*]
Hangzhou Dianzi University
246050120@hdu.edu.cn, chenzhichao_ai@163.com,
lqyan18@fudan.edu.cn, yycheng@hdu.edu.cn, fangli.guan@hdu.edu.cn, lipan@ieee.org

Abstract

The computational power required for training deep learning models has been skyrocketing in the past decade as they scale with big data, and has become a very expensive and scarce resource. Therefore, distributed training, which can leverage distributed available computational power, is vital for efficient large-scale model training. However, most previous distributed training frameworks like DDP and DeepSpeed are primarily designed for co-located clusters under homogeneous computing and communication conditions, and hence cannot account for geo-distributed clusters with both computing and communication heterogeneity. To address this challenge, we develop a new data parallel based distributed training framework called Co-Adaptive Data Parallelism (C-ADP). First, we consider a data owner and parameter server that distributes data to and coordinates the collaborative learning across all the computing devices. We employ local training and delayed parameter synchronization to reduce communication costs. Second, we formulate a data parallel scheduling optimization problem to minimize the training time by optimizing data distribution. Third, we devise an efficient algorithm to solve this scheduling problem, and formally prove that the obtained solution is optimal in the asymptotic sense. Experiments on the ImageNet100 dataset demonstrate that C-ADP achieves fast convergence in heterogeneous distributed training environments. Compared to Distributed Data Parallel (DDP) and DeepSpeed, C-ADP achieves 21.6 times and 26.3 times improvements in FLOPS, respectively, and a reduction in training time of about 72% and 47%, respectively.

1 Introduction

The rapid expansion of Internet applications has led to an exponential increase in global data, with recent data generation surpassing that of the entire previous history [Yi *et al.*, 2020]. The surge of big data stimulates the rapid expansion of deep learning models, ranging from simple neural networks to Large Language Models (LLMs) containing hundreds of billions of parameters, such as the GPT family [Brown *et al.*, 2020] and Transformer [Wang *et al.*, 2022], thereby placing an increasingly urgent demand on efficient training of large-scale learning models.

In order to cope with this challenge, distributed training has emerged. Its core concept is to leverage distributed nodes to decompose and distribute computational tasks for collaborative completion, effectively overcoming the limitations of single-machine processing power. Data parallelism, a key form of distributed computing, is widely used in training large-scale learning models [Li *et al.*, 2020; Verbraeken *et al.*, 2020]. It efficiently trains complex models by splitting training data and distributing it across computing nodes. In Computer Vision (CV), data parallelism enables efficient training for tasks such as image classification and object detection, and improves model accuracy by extracting rich visual features. In Natural Language Processing (NLP), data parallelism ensures the processing of large-scale text data, helping models capture intricate semantic and syntactic relationships, which drives continuous advancements in NLP technologies.

Despite the considerable benefits of data parallelism in enhancing computational efficiency, its practical implementation encounters numerous challenges. First, distributed computing could introduce large communication overhead. In the data distribution phase, the transmission of large volumes of training data is limited by network bandwidth and latency, while in the training process, frequent synchronization of gradients or parameters further increases the communication cost. Second, computing device heterogeneity can lead to ineffective data distribution and uneven loading due to uniform data distribution assumed by popular distributed training mechanisms like Distributed Data Parallelism (DDP) [Li *et al.*, 2020]. Although frameworks such as DeepSpeed [Ren *et al.*, 2021; Rasley *et al.*, 2020] have improved memory management, they are still deficient in balanced data allocation. Third, most distributed training frameworks such as DDP and DeepSpeed are primarily designed for co-located clusters with homogeneous network conditions, and hence cannot account for geo-distributed clusters with heterogeneous network environment over the Interne.

To address these challenges, we develop a new **C**o-**A**daptive **D**ata **P**arallelism (**C-ADP**) framework by optimiz-

[*] Corresponding authors: Pan Li, Liqi Yan.

ing data parallel scheduling while taking into account computing and communication heterogeneity in a geo-distributed cluster. Specifically, C-ADP enables more efficient and robust distributed training through the following designs. (1) Delayed parameter synchronization. By employing local training and delayed parameter synchronization at the conclusion of each epoch instead of each batch, we substantially reduce communication overhead. (2) Data parallel scheduling formulation. We formulate a data parallel scheduling optimization problem to minimize the training time by optimizing data distribution. (3) An efficient algorithm. We develop a very efficient algorithm to solve the optimization problem, and formally prove that the obtained solution is optimal in the asymptotic sense. Through strategic data distribution, we ensure balanced loading across computing devices with heterogeneous capabilities and significantly reduced communication overhead, thus accelerating the training process and enhancing overall efficiency.

In this paper, our main contributions are threefold:

- We propose a new data parallel based distributed training framework, named Co-Adaptive Data Parallelism (C-ADP), for a geo-distributed cluster with heterogeneous computing and communication environments.
- We formulate data parallel scheduling as an optimization problem and develop an efficient algorithm that is able to efficiently and optimally determine the amount of data samples to be allocated to different computing devices.
- Experiments on the ImageNet100 dataset show that, compared with DDP and DeepSpeed, our C-ADP framework achieves 21.6 times and 26.3 times improvements in FLOPS, respectively, and a reduction in trainning time of about 72% and 47%, respectively.

2 Related Work

Distributed Training Architectures: There have been a few distributed training frameworks such as DDP [Li *et al.*, 2020], DeepSpeed [Rasley *et al.*, 2020], Megatron-LM [Shoeybi *et al.*, 2020]. Data parallelism [Zhao *et al.*, 2023], pipeline parallelism [Narayanan *et al.*, 2019], tensor parallelism [Shazeer *et al.*, 2018] are key methods for parallelizing operations during the training process. In data parallelism, a global model is replicated across a cluster of computing devices, each of which is allocated a subset of data, and is collaboratively trained by them. Synchronous and asynchronous Stochastic Gradient Descent (SGD) are commonly used for convergence and scalability, leveraging parameter server or ring-allreduce [Yu *et al.*, 2022] architectures. The parameter server architecture uses a centralized server for global parameters, while ring-allreduce employs a decentralized approach. Pipeline parallelism divides the model layers into stages and distributes them across different computing devices [Narayanan *et al.*, 2019]. Tensor parallelism shards the model's parameters and computations along tensor dimensions and distributes the computations of a single layer, e.g., matrix multiplications across devices. Frameworks like DDP [Li *et al.*, 2020], Horovod [Sergeev and Balso, 2018] use pure data parallelism. Those like DAPPLE [Fan *et al.*, 2021], PipeDream [Narayanan *et al.*, 2019] combines data and pipeline parallelism. and Megatron-LM [Shoeybi *et al.*, 2020], AMP [Li *et al.*, 2022] integrate all three types of parallelism. In this paper, we focus on data parallelism since it has excellent scalability and usually moderate communication cost compared with pipeline and tensor parallelism.

Symbol	Definition
N	The number of computing devices
D	The total amount of data samples (the entire dataset)
D_i	The total amount of data sample assigned to device i ($i \in [1, N]$)
b	The size of each sample
G_i	The total memory size of device i
$B_{j,i}$	The transmission bandwidth from device j to device i
$batchsize(i)$	The batch size used by device i in each iteration
S_m	The size of occupied memory samples
ΔS_i	Activation memory consumption increased due to adding one more data sample
$\alpha_{j,i}$	Communication latency
M_m	The size of model parameters
num	The number of epochs at all devices

Table 1: Key notations in this paper.

Scheduling Optimization in Heterogeneous Environments: A few mechanisms [Jayaram Subramanya *et al.*, 2023; Mo *et al.*, 2024; Unger *et al.*, 2022] have been developed to optimize cluster-level scheduling. In particular, [Jayaram Subramanya *et al.*, 2023; Chen *et al.*, 2022] optimize the scheduling of multiple jobs, e.g., by increasing data parallelism, on heterogeneous GPU clusters. [Yang *et al.*, 2024] intelligently assigns different computational tasks in model training within heterogeneous environments based on the characteristics of the NICs to which the GPU devices are connected. [Yuan *et al.*, 2022] minimizes communication costs in heterogeneous environments. [Liu *et al.*, 2023] designs a reinforcement learning-based scheduling method to assign each layer to an appropriate type of computing resource, minimizing cost while ensuring throughput limits based on a provisioning method. Previous scheduling optimization mechanisms mostly only considers either computing or communication heterogeneity. In this paper, we address both computing and communication heterogeneity in data parallelism.

3 System Model and Problem Formulation

We consider a geo-distributed system with a data owner and parameter server (PS) and N computing devices as shown in Figure 1. After finding the optimal distributed training schedule, with will be delated later, the PS sends the data subset D_i ($1 \leq i \leq N$) to each participating computing devices i. Each device i performs local training on the received data subset D_i using stochastic gradient descent (SGD), and then uploads the locally updated model parameters w_i^{t+1} based on the cur-

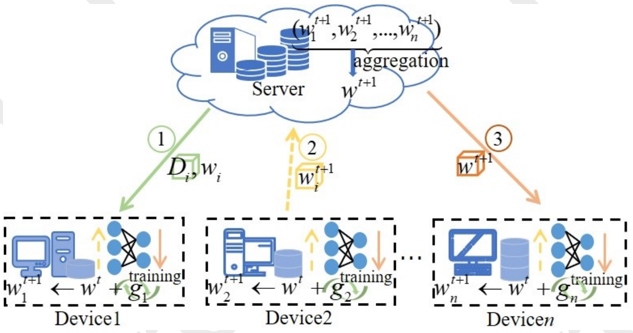

$(w_1^{t+1}, w_2^{t+1}, ..., w_n^{t+1})$
aggregation
Server
w^{t+1}
1
D_i, w_i
2
w_i^{t+1}
3
w^{t+1}
training
$w_1^{t+1} \leftarrow w^t + g_1$
Device1
$w_2^{t+1} \leftarrow w^t + g_2$
Device2
...
$w_n^{t+1} \leftarrow w^t + g_n$
Devicen

Figure 1: The workflow of the proposed C-ADP framework.

rent global model w^t to the central server. The PS performs weighted averaging based on the local model parameters of all devices to update the global model w^{t+1} distributes it to all devices, i.e.,

$$w^{t+1} = \sum_{i=1}^{N} \frac{D_i}{D} w_i^{t+1},$$

where D is the total amount of data samples. Then, we have

$$\sum_{i=1}^{N} D_i = D, \tag{1}$$

where $D_i \in \mathbb{N}_0$, we assume that all data samples have the same size of b bits.

In the realm of distributed training, we focus solely on data parallelism, where our primary objective is to minimize the total time consumed by the entire training process encompassing multiple epochs. This total time is composed of three crucial components: data allocation time, computation time and communication time. Some important notations are summarized in Table 1.

3.1 Data Allocation Time

Denote by $B_{0,i}$ the bandwidth between device 0 and device i. The communication time for device i to receive its allocated D_i data samples, denoted by $T_{\text{send},i}$, is

$$T_{\text{send},i} = \frac{bD_i}{B_{0,i}}, \tag{2}$$

3.2 Computation Time

During distributed training, the computational time of each device is related to the amount of data allocated to it, i.e., the larger the dataset, the longer the computation time. Denote the maximum parallelism of device i as $P_{\text{max}}(i)$, i.e., the maximum number of samples that a device can process concurrently. Then, we can get

$$P_{\text{max}}(i) = \frac{G_i - S_m}{\Delta S_i}, \tag{3}$$

which needs to be no less than the batchsize at device i, denoted by $batchsize(i)$, i.e.,

$$P_{\text{max}}(i) \geq batchsize(i), \tag{4}$$

Then, the computation time at device i during one epoch, denoted by $T_{\text{comp},i}$, can be obtained by

$$T_{\text{comp},i} = \left\lceil \frac{D_i}{batchsize(i)} \right\rceil \times T_i^{\text{batch}}, \tag{5}$$

where T_i^{batch} denotes the time required by device i to process a single batch of data, including both forward propagation and backward propagation computations.

3.3 Communication Time

Each computing device i receives the current global model parameters from the PS, updates its local model using its allocated training data, and sends the updated model parameters to the PS at the end of each epoch. We denote by M_m is the size of the model parameters. Thus, the communication cost of device i for data parallelism in one epoch, denoted by $T_{\text{comm},i}$ can be calculated as follows:

$$T_{\text{comm},i} = 2\left(\alpha_{0,i} + \frac{M_m}{B_{0,i}}\right). \tag{6}$$

3.4 Data Parallel Scheduling Optimization

To minimize the total time for the above data parallel training process, we aim to optimize the data allocation D_i $(1 \leq i \leq N)$. Consequently, the data parallel scheduling optimization problem can be formulated as:

$$\begin{aligned} &\min_{D_1,\cdots,D_N} \left(\max_{i\in[1,N]} T_{\text{send},i} + \sum_{epoch=1}^{num} \max_{i\in[1,N]} \left(T_{\text{comp},i} + T_{\text{comm},i}\right)\right), \\ &\qquad \text{subject to} \quad (1)(4), \end{aligned} \tag{7}$$

4 An Efficient Data Parallel Scheduling

4.1 Optimality Analysis

To simplify notations, we let $\boldsymbol{x} = \{D_1, D_2, ..., D_N\}$. We rewrite the optimization problem formulated in equation (7) into the following:

$$\begin{aligned} \text{F}(\boldsymbol{x}) &= \max \boldsymbol{f}(\boldsymbol{x}) + num \cdot \max(\boldsymbol{g}(\boldsymbol{x}) + \mathbf{C}), \\ &\min_{\boldsymbol{x}} \text{F}(\boldsymbol{x}), \end{aligned} \tag{8}$$

where

$$\begin{cases} \boldsymbol{f}(\boldsymbol{x}) = b \times (\mathbf{V}^{-1} \times \boldsymbol{x}), \quad \mathbf{V} = \begin{pmatrix} B_{0,1} & 0 & \cdots & 0 \\ 0 & B_{0,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{0,N} \end{pmatrix}, \\ \boldsymbol{g}(\boldsymbol{x}) = \mathbf{T} \times \lceil \mathbf{B}^{-1} \times \boldsymbol{x} \rceil + \mathbf{C}, \\ \mathbf{T} = \begin{pmatrix} T_1^{batch} & 0 & \cdots & 0 \\ 0 & T_2^{batch} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & T_N^{batch} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} batchsize_1 & 0 & \cdots & 0 \\ 0 & batchsize_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & batchsize_N \end{pmatrix}, \\ \mathbf{C} = 2(\boldsymbol{\alpha} + \mathbf{B}^{-1} \cdot \mathbf{M}), \quad \boldsymbol{\alpha} = \begin{pmatrix} \alpha_{0,1} \\ \alpha_{0,2} \\ \cdots \\ \alpha_{0,N} \end{pmatrix}, \\ \boldsymbol{M} = \begin{pmatrix} M_m \\ M_m \\ \cdots \\ M_m \end{pmatrix}_{N\times 1} \quad \sum_{i=1}^{n} x_i = D, \\ x_i \in \mathbb{N}_0. \end{cases}$$

Note that is $\mathbf{C}$ independent of the variable $\boldsymbol{x}$. It dependents on communication bandwidth $\mathbf{B}$ and model size M_m, and hence it is a constant.

Lemma 1. *$x \geq \mathbf{0}$, we have $\lim_{p\to\infty} \|\boldsymbol{x}\|_p = \max\ \boldsymbol{x}$.*

Proof. Let $\hat{x} = \max\ \boldsymbol{x}$. Then, we have

$$\lim_{p\to\infty} (\sum_{k=1}^{N} x_k^p)^{\frac{1}{p}} \leq \lim_{p\to\infty} (N\hat{x}^p)^{\frac{1}{p}} = \lim_{p\to\infty} N^{\frac{1}{p}}\hat{x} = \hat{x},$$

$$\lim_{p\to\infty} (\sum_{k=1}^{N} x_k^p)^{\frac{1}{p}} \geq \lim_{p\to\infty} (\hat{x}^p)^{\frac{1}{p}} = \hat{x}.$$

Therefore, we can find $\lim_{p\to\infty} \|\boldsymbol{x}\|_p = \lim_{p\to\infty} (\sum_{k=1}^{N} x_k^p)^{\frac{1}{p}} = \max\ \boldsymbol{x}$. □

In fact, when $p = 100$, the error between $\|\boldsymbol{x}\|_p$ and $\max\ \boldsymbol{x}$ is already sufficiently small, as shown in Figure 2.

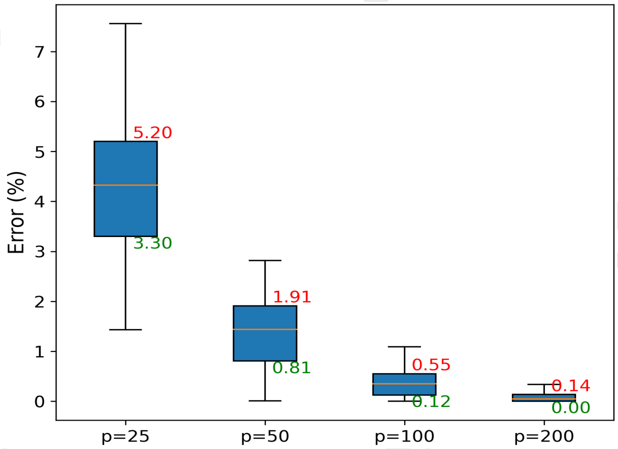


Figure 2: Error between $\|\boldsymbol{x}\|_p$ and $\max\ \boldsymbol{x}$ for different values of p.

Moreover, we can also prove that $\|\boldsymbol{x}\|_p$ has the same subadditivity and homogeneity as $\max\ \boldsymbol{x}$.

Lemma 2. *For any $\boldsymbol{x}, \boldsymbol{y} \in \{(x_1, x_2, ..., x_N) | \sum_{i=1}^{N} x_i = D, x_i \geq 0\}$, and $\lambda \in R$, we have $\|\boldsymbol{x} + \boldsymbol{y}\|_p \leq \|\boldsymbol{x}\|_p + \|\boldsymbol{y}\|_p$, and $||\lambda\boldsymbol{x}||_p = |\lambda| \cdot ||\boldsymbol{x}||_p$.*

Proof. According to the $Minkowski$ inequality, we get

$$(\sum_{k=1}^{N} |x_k + y_k|^p)^{\frac{1}{p}} \leq (\sum_{k=1}^{N} |x_k|^p)^{\frac{1}{p}} + (\sum_{k=1}^{N} |y_k|^p)^{\frac{1}{p}}.$$

Therefore, we get

$$\begin{aligned} \|\boldsymbol{x} + \boldsymbol{y}\|_p &= (\sum_{k=1}^{N} (x_k + y_k)^p)^{\frac{1}{p}} \\ &\leq (\sum_{k=1}^{N} x_k^p)^{\frac{1}{p}} + (\sum_{k=1}^{N} y_k^p)^{\frac{1}{p}} \\ &= \|\boldsymbol{x}\|_p + \|\boldsymbol{y}\|_p. \end{aligned}$$

and

$$||\lambda\boldsymbol{x}||_p = (\sum_{k=1}^{N} |\lambda\boldsymbol{x_k}|^p)^{\frac{1}{p}} = |\lambda|(\sum_{k=1}^{N} (\boldsymbol{x_k})^p)^{\frac{1}{p}} = |\lambda| \cdot ||\boldsymbol{x}||_p.$$

□

Consequently, according to Lemma 1, when p is large, the objective function in equation (8) can be rewritten into

$$\begin{aligned} \Phi(\boldsymbol{x}) &= \|\boldsymbol{f}(\boldsymbol{x})\|_p + num \cdot\ \|\boldsymbol{g}(\boldsymbol{x}) + \mathbf{C}\|_p \\ &= \left\|b \cdot \mathbf{V}^{-1}\boldsymbol{x}\right\|_p + num \cdot \left\|\mathbf{T}\lceil\mathbf{B}^{-1}\boldsymbol{x}\rceil + \mathbf{C}\right\|_p, \end{aligned} \tag{9}$$

whose feasible domain is

$$\chi = \{(x_1, x_2, ..., x_N) | \sum_{i=1}^{N} x_i = D, x_i \in \mathbb{N}_0\}.$$

We then define a relaxed function

$$\widetilde{\Phi}(\boldsymbol{x}) = \left\|b \cdot \mathbf{V}^{-1}\boldsymbol{x}\right\|_p + num \cdot \left\|\mathbf{T}\mathbf{B}^{-1}\boldsymbol{x} + \mathbf{C}\right\|_p \tag{10}$$

whose feasible domain is

$$\chi' = \{(x_1, x_2, ..., x_N) | \sum_{i=1}^{N} x_i = D, x_i \geq 0\}.$$

We arrive at the following theorem, the proofs of which can be found in the appendices.

Theorem 1. *The relaxed function equation (10) is a convex function on its feasible domain.*

Lemma 3. *For any $\boldsymbol{x} \in R^N$ and $\boldsymbol{x} \geq 0$, we have $\|\lceil\boldsymbol{x}\rceil - \boldsymbol{x}\|_p < N^{\frac{1}{p}}$.*

Proof. For any element $x_k \in \boldsymbol{x}$, we have $|\lceil x_k\rceil - x_k| < 1$. Thus, we can obtain

$$\begin{aligned} \|\lceil\boldsymbol{x}\rceil - \boldsymbol{x}\|_p &= (\sum_{k=1}^{N} (\lceil x_k\rceil - x_k)^p)^{\frac{1}{p}} \\ &\leq N^{\frac{1}{p}}. \end{aligned}$$

□

Theorem 2. *Given $\boldsymbol{x} \in \chi'$, for any $\hat{\boldsymbol{x}} \in \chi$ that satisfies*

$$\|\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|_p \leq \gamma,$$

where $\gamma \geq N^{\frac{1}{p}}$, we have

$$|\Phi(\hat{\boldsymbol{x}}) - \widetilde{\Phi}(\tilde{\boldsymbol{x}})| \leq M_\gamma$$

where $M_\gamma = b\gamma \cdot \left\|\boldsymbol{V}^{-1}\right\|_p + num \cdot \|\boldsymbol{T}\|_p (N^{\frac{1}{p}} + \gamma \left\|\boldsymbol{B}^{-1}\right\|_p)$.

According to Theorem 2, we first find the optimal solution $\tilde{\boldsymbol{x}}^*$ to the relaxed equation (10). Then, we can find $\hat{\boldsymbol{x}} \in \chi$, in the neighborhood of $\tilde{\boldsymbol{x}}^*$, i.e., $U(\tilde{\boldsymbol{x}}^*, \gamma)$, $\gamma \geq N^{\frac{1}{p}}$, such that $|\Phi(\hat{\boldsymbol{x}}) - \widetilde{\Phi}(\tilde{\boldsymbol{x}}^*)| \leq M_\gamma$. Note that M_γ is independent of the total amount of data samples D. Therefore, for the optimal solution $\hat{\boldsymbol{x}}^*$ to the original equation (9), we have $\widetilde{\Phi}(\tilde{\boldsymbol{x}}^*) \leq \Phi(\hat{\boldsymbol{x}}^*) \leq \Phi(\hat{\boldsymbol{x}})$, and hence, $\Phi(\hat{\boldsymbol{x}}) - \Phi(\hat{\boldsymbol{x}}^*) \leq M_\gamma$. Since $\Phi(\hat{\boldsymbol{x}}^*)$ is positively correlated to D, we can have

$$\lim_{D\to\infty} \frac{\Phi(\hat{\boldsymbol{x}}) - \Phi(\hat{\boldsymbol{x}}^*)}{\Phi(\hat{\boldsymbol{x}}^*)} = 0,$$

i.e., we can find a solution that is optimal in the asymptotic sense when there are a sufficiently large number of training data samples.

Algorithm 1 Our Data Parallel Scheduling Algorithm

```
Input: $S$, $n$, $\rho$, $\mu^{(0)}$, $\mu_{\text{decay}}$, $\lambda^{(0)}$, $\varepsilon$, $\eta^{(0)}$, $\eta_{\text{decay}}$, $\psi$, $\zeta$
Output: $\hat{\boldsymbol{x}}$
Let $\boldsymbol{x} \leftarrow \frac{S}{n}\mathbf{1}$ , $k \leftarrow 0$ ,$\mu \leftarrow \mu^{(0)}$ ,$\lambda \leftarrow \lambda^{(0)}$ , $\eta \leftarrow \eta^{(0)}$ .
while $k < \psi$ do
    $i \leftarrow 0$
    while $i < \zeta$ do
        Compute gradients $\nabla_{\boldsymbol{x}}\mathcal{L}_\rho(\boldsymbol{x}^{(k,i)}, \lambda^{(k)}, \mu^{(k)})$
        Update
        $\boldsymbol{x}^{(k,i+1)} = \boldsymbol{x}^{(k,i)} - \eta^{(k)} \cdot \nabla_{\boldsymbol{x}}\mathcal{L}_\rho(\boldsymbol{x}^{(k,i)}, \lambda^{(k)}, \mu^{(k)})$
        if $\|\nabla\mathcal{L}\|_2 \leq \varepsilon$ then
            break
        end if
        $i \leftarrow i + 1$
    end while
    Update
        $$\lambda^{(k+1)} \leftarrow \lambda^{(k)} + \rho\left(\sum_j x_j^{(k+1,0)} - S\right)$$
    Update
        $$\mu^{(k+1)} \leftarrow \mu_{decay} \cdot \mu^{(k)}$$
    if $\|\nabla\mathcal{L}\|_2 \leq \varepsilon$ then
        break
    end if
    $k \leftarrow k + 1$
end while
    $\tilde{\boldsymbol{x}}^* \leftarrow \boldsymbol{x}$
    $\hat{\boldsymbol{x}} \leftarrow$ Round $(\tilde{\boldsymbol{x}}^*)$
return $\hat{\boldsymbol{x}}$
```

4.2 Algorithm Design

Based on the analysis in Section 4.1, we design an algorithm that first solves for the optimal solution $\tilde{\boldsymbol{x}}^*$ to equation (10), and then finds the solution to the original equation (9), i.e., $\hat{\boldsymbol{x}}^*$ by rounding each element of $\tilde{\boldsymbol{x}}^*$ to an integer. According to Theorem 1, we are able to find a globally optimal solution to the relaxed equation (10). So we construct an augmented Lagrangian function as follows:

$$\mathcal{L}_\rho(\boldsymbol{x}, \lambda, \mu) = \widetilde{\Phi}(\boldsymbol{x}) + \lambda\left(\sum_{k=1}^{N} x_k - D\right) + \frac{\rho}{2}\left(\sum_{k=1}^{N} x_k - D\right)^2 - \mu\sum_{k=1}^{n}\ln(x_k)$$

Where λ is the Lagrange multiplier for the reciprocal constraint, $\rho > 0$ is the coefficient of the quadratic penalty term, and $\mu > 0$ is the coefficient of the logarithmic barrier used to ensure that $x_i \geq 0$. We detail our data parallel scheduling algorithm in Algorithm 1.

4.3 Computational Complexity Analysis

The inner loop of Algorithm 1 involves computing the gradients and diagonal matrix multiplication. Since the inner loop is executed at most ζ times, the computational complexity is $O(\zeta N)$. The outer loop is executed at most ψ times. Therefore, the computational complexity of our data parallel scheduling algorithm is $O(\psi\zeta N)$, where N is the number of computing devices in the system.

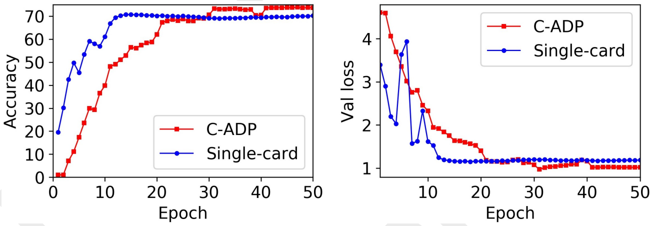


Figure 3: Changes in test accuracy and loss under the conditions of C-ADP and single-card training.

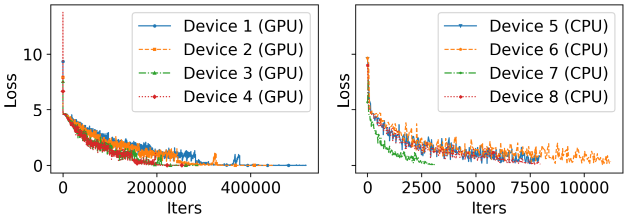


Figure 4: Training loss curves of Wide-ResNet101 on heterogeneous devices.

5 Experiment Results

5.1 Dataset and Learning Model

Dataset: The experiments were carried out using the ImageNet100 dataset, a subset of the ImageNet dataset. ImageNet100 consists of 100 classes, particularly 100,000 training samples and 10,000 test samples, making it a balanced and diverse dataset for evaluating model performance.

Models: To evaluate C-ADP, Wide - ResNet101 was chosen and validated on ImageNet100. Wide-ResNet101, engineered to process intricate image features, incurs a computational cost of 22.8 GFLOPs.

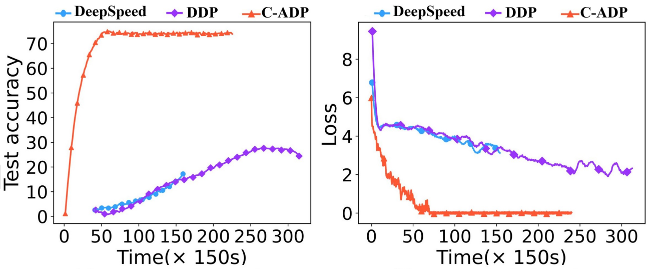
5.2 Experimental Environment and Metrics

Computing Devices: To account for heterogeneous environments, the hardware setup is as follows. In the Zhongwei area, rank 0 as a data owner and parameter server handles data transfer and parameter synchronization. Ranks 1 - 4 use RTX3090 GPUs with 24GB VRAM. Ranks 5 - 6 are 4 core CPUs with 8GB memory. Rank7 is a 16 core CPU with 32GB VRAM, and rank8 is an 8 core CPU with 16GB VRAM. To emulate real-world constraints, we simulate network heterogeneity among these devices by controlling the available bandwidth between devices.

Hyperparameter Configuration: For all GPUs, the batch size is set to 64 due to its high computational complexity, to ensure stable training within the GPU's memory capacity. For all CPUs, the batch size is set to 4, considering their limited

	Acc/%		F1/%		Precision/%		Recall/%		Test loss	
Epoch	C-ADP	Single-card	C-ADP	Single-card	C-ADP	Single-card	C-ADP	Single-card	C-ADP	Single-card
1	1.00	19.53	0.02	17.13	0.01	22.71	1.00	19.53	4.606	3.3924
10	39.81	61.08	38.67	60.61	43.43	64.95	39.81	61.00	2.328	1.6175
20	62.04	70.17	61.7	69.89	62	70.31	64.62	70.17	1.400	1.1625
30	70.56	69.46	70.22	68.81	71.53	69.2	70.55	69.12	1.074	1.2012
40	70.49	70.16	70.23	69.99	71.78	70.55	70.49	70.13	1.159	1.1947
50	73.67	71.02	73.46	70.97	73.75	71.21	73.66	71.05	1.018	1.2039

Table 2: Accuracy metrics and validation loss results of our method and single-card training under the condition of interval epochs.



(a) Curve depicting the relationship between model accuracy and training time

(b) Curve depicting the relationship between training loss and training time

Figure 5: Results of accuracy and loss of different training frameworks in the time dimension.

Method	Epoch	Acc/%	T_{send}/ s	T_{train}/ s	T_{total}/ s
DDP	7	27.44	1411.68	47214.33	48626.02
Deepspeed	4	16.94	1398.46	23882.89	25281.35
C-ADP	50	74.36	431.82	12915.28	13347.11

Table 3: Training results of different training frameworks.

computing power. All settings are designed to keep operations within the device's memory and computational limits. The maximum number of epochs is set to 50. The initial learning rate is 0.01. To improve convergence, we reduce the learning rate by half after every 10 epochs.

Baselines: To ensure comprehensive performance comparison, two widely-used and efficient frameworks are selected as baselines, as they represent state-of-the-art solutions to distributed deep learning training:

- **Deepspeed**: an open source library developed by Microsoft to accelerate the training of deep learning models. It provides an efficient training framework that supports distributed training, model parallelism, and data parallelism.
- **DDP (Distributed Data Parallel):** a widely-adopted feature in PyTorch. DDP provides robust support for distributed training by replicating models across multiple devices and synchronizing gradients.

Metrics: The following metrics are adopted to evaluate the performance of C-ADP and the baselines comprehensively:

- Accuracy Metrics: standard metrics including Accuracy (Acc), F1-Score (F1), and Precision to assess the quality of the trained model.
- T_{send}: the data transmission time required for computing devices to receive datasets from the rank 0 data owner and parameter server.
- T_{training}: the end-to-end training time, which includes the forward propagation, backward propagation, and parameter synchronization.
- T_{total}: The total time for completing a fixed number of epochs on a device, calculated as $T_{\text{total}} = T_{\text{send}} + T_{\text{training}}$.

5.3 Model Convergence in Heterogeneous Environments

Verification of Accuracy Metrics: We trains the computationally intensive WideResNet101 on ImageNet100. To address the frequent out-of-memory issues when using a single 3090 card, we employ a more powerful A800 card for single-card training and compare its performance with the proposed C-ADP framework. Training accuracy, validation loss, and final metrics are shown in Figure 3 and Table 2.

Loss Convergence Validation: To validate the convergence properties of C-ADP in a heterogeneous environment, we show the training loss curves of all computing devices in Figure 4. As can be clearly observed, C-ADP achieves successful convergence across all 8 devices, with the loss function steadily decreasing and low loss values reached. Additionally, a detailed examination reveals that the scheduling method employed effectively allocates data based on the performance disparities among devices, enabling efficient handling of data processing tasks across devices with varying capabilities, without requiring the same number of iterations.

Result Analysis: Early in training, single-card training outperforms C-ADP. But as the number of training rounds grows, C-ADP takes the lead. At the 30th round, C-ADP accuracy reached 70.56%, exceeding the single-card's accuracy of 69.46%, and increases to 73.67% at the 50th round. F1-score, precision, and recall follows similar trends. The test loss is lower in the case of single-card training initially, but lower for C-ADP later, suggesting better late-stage performance. Despite slow convergence due to low-frequency parameter synchronization, C-ADP achieves convergence across all devices. It also outperforms single-card training in key metrics like accuracy and F1-score later on, proving effectiveness for improving model accuracy. Its data-parallel strategy can leverage distributed nodes, compensating for single-device limitations, which is crucial for large-scale training.

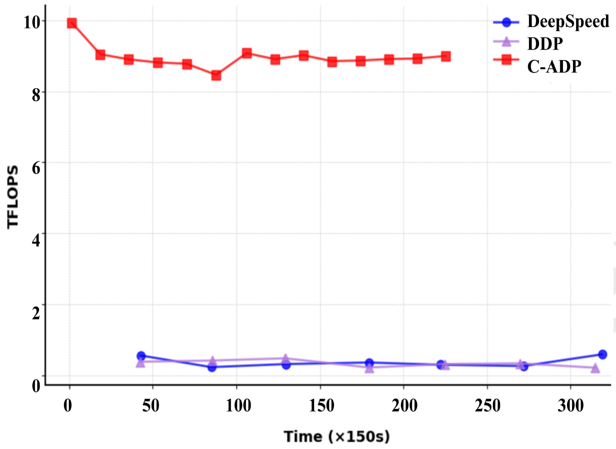


Figure 6: Comparison results of FLOPS of different frameworks during the training process.

5.4 Comparison with Other Data Parallel Frameworks

Training Time: Figure 5 compares the training processes of the three frameworks, measured in time. C-ADP optimizes data allocation, improves the throughput of large-scale models, and completes the training in the shortest time. DDP fails to converge for a long time, and its accuracy decreases in the later stage. Deepspeed, due to frequent parameter updates, incurs high communication costs during the training of large-scale models, which slows down the training speed. Through comprehensive comparison, C-ADP outperforms the data-parallel frameworks DDP and Deepspeed in terms of training speed. It can converge smoothly and quickly and complete the model training.

Comparison of FLOPS: As illustrated in Figure 6, during the evaluation, the device FLOPS under the C-ADP training method consistently stabilizes at 8.95 TFLOPS, achieving 21.6 times improvement over DDP and 26.3 times improvement over DeepSpeed. In contrast, other methods exhibites significant performance fluctuations due to the impact of heterogeneous computing and communication capabilities of all computing devices, leading to reduced resource utilization efficiency. By optimizing the data allocation strategy and reducing the parameter synchronization frequency, C-ADP effectively mitigates the impact of communication bottlenecks on training performance. This approach not only significantly improves computational efficiency but also ensures system robustness and resource utilization, demonstrating its advantages in large-scale distributed training.

Quantization Result Comparison: Table 3 compares the performance of the three distributed training frameworks, including DDP, Deepspeed, and C-ADP. C-ADP exhibits distinct advantages. Due to its adaptive data allocation strategy, C-ADP only requires a data transfer time of 431.82s, much less than the 1411.68s by DDP and 1398.46s by Deepspeed. The C-ADP framework achieves even loading across all devices by fully addressing heterogeneous computing and communication constraints in the system.

Result Analysis: Our C-ADP framework focuses on finding the optimal data distribution to all the computing devices in distributed training. As shown in Table 3, C-ADP converges after 12915.28s and acheives an accuracy of 74.36%, while other two frameworks. As a result, C-ADP can achieve convergence more rapidly during the training process. In summary, C-ADP significantly reduces the total time for distributed training and achieves a reduction in trainning time of about 72% and 47%, respectively, compared with DDP and DeepSpeed, making it a more efficient and reliable option for large-scale deep learning model training.

6 Conclusion

In this paper, we explore the opportunity to train deep learning models in a distributed manner with computing devices connected in a heterogeneous environment. We have designed an efficient and effective data parallel training framework called C-ADP that can find optimal data distribution in the asymptotic sense. Experiments on Wide-ResNet101 with the ImageNet100 dataset show that C-ADP achieves excellent convergence performance in heterogeneous environments, reaching 73.67% accuracy at the 50th round of training and outperforming single-device training in key metrics such as F1- score. Compared with DDP and DeepSpeed, C-ADP achieves 21.6 times and 26.3 times improvements in FLOPS, respectively, and a reduction in training time of about 72% and 47%, respectively.

A Proof of Theorem 1

Proof. Consider an optimization domain $D^N = \{(x_1, x_2, ..., x_N) | \sum_{i=1}^{N} x_i \le D, x_i \ge 0\}$. Obviously, this domain is convex. Thus, according to Lemma 2, for any $\boldsymbol{x}, \boldsymbol{y} \in D^N, \lambda \in (0, 1)$, the first component on the right-hand side of equation (10) is

$$\begin{aligned}
&\left\|b \cdot \mathbf{V}^{-1}(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y})\right\|_p \\
&\le \left\|b \cdot \mathbf{V}^{-1}\lambda \boldsymbol{x}\right\|_p + \left\|b \cdot \mathbf{V}^{-1}(1-\lambda)\boldsymbol{y}\right\|_p \\
&= \lambda\left\|b \cdot \mathbf{V}^{-1}\boldsymbol{x}\right\|_p + (1-\lambda)\left\|b \cdot \mathbf{V}^{-1}\boldsymbol{y}\right\|_p .
\end{aligned}$$

□

which is a convex function. Similarly, we can easily show that the second component on the right-hand side of equation (10) is a convex function as well. Therefore, we can have that $\widetilde{\Phi}(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \le \lambda \widetilde{\Phi}(\boldsymbol{x}) + (1-\lambda)\widetilde{\Phi}(\boldsymbol{y})$, and the function $\widetilde{\Phi}(\boldsymbol{x})$ is convex on the domain D^N. Consequently, $\widetilde{\Phi}(\boldsymbol{x})$ is also a convex function on the boundary $\partial D^N = \chi' = \{(x_1, x_2, ..., x_N) | \sum_{i=1}^{N} x_i = D, x_i \ge 0\}$ of this domain.

B Proof of Theorem 2

Proof. Let $\boldsymbol{\delta} = \hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}$, where $\hat{\boldsymbol{x}} \in \chi$, and $\tilde{\boldsymbol{x}} \in \chi'$. Then, we have $\|\boldsymbol{\delta}\|_p \le \gamma$. According to equations (9) and (10), we can get

$$\begin{aligned}\left|\Phi(\hat{\boldsymbol{x}})-\widetilde{\Phi}(\tilde{\boldsymbol{x}})\right| &= \Big| \left\|b\cdot\mathbf{V}^{-1}\hat{\boldsymbol{x}}\right\|_p - \left\|b\cdot\mathbf{V}^{-1}\tilde{\boldsymbol{x}}\right\|_p \\ &\quad + num\cdot\Big(\left\|\mathbf{T}\lceil\mathbf{B}^{-1}\hat{\boldsymbol{x}}\rceil+\mathbf{C}\right\|_p - \left\|\mathbf{T}\mathbf{B}^{-1}\tilde{\boldsymbol{x}}+\mathbf{C}\right\|_p\Big)\Big| \\ &\le \left|\left\|b\cdot\mathbf{V}^{-1}\hat{\boldsymbol{x}}\right\|_p - \left\|b\cdot\mathbf{V}^{-1}\tilde{\boldsymbol{x}}\right\|_p\right| \\ &\quad + num\cdot\left|\left\|\mathbf{T}\lceil\mathbf{B}^{-1}\hat{\boldsymbol{x}}\rceil+\mathbf{C}\right\|_p - \left\|\mathbf{T}\mathbf{B}^{-1}\tilde{\boldsymbol{x}}+\mathbf{C}\right\|_p\right|,\end{aligned} \tag{11}$$

For the first component on the right-hand side of the above inequality, we have

$$\begin{aligned}\left\|b\cdot\mathbf{V}^{-1}\hat{\boldsymbol{x}}\right\|_p &= \left\|b\cdot\mathbf{V}^{-1}(\tilde{\boldsymbol{x}}+\boldsymbol{\delta})\right\|_p \\ &\le \left\|b\cdot\mathbf{V}^{-1}\tilde{\boldsymbol{x}}\right\|_p + \left\|b\cdot\mathbf{V}^{-1}\boldsymbol{\delta}\right\|_p \\ &\overset{\text{(a)}}{\le} \left\|b\cdot\mathbf{V}^{-1}\tilde{\boldsymbol{x}}\right\|_p + b\cdot\left\|\mathbf{V}^{-1}\right\|_p\left\|\boldsymbol{\delta}\right\|_p \\ &\le \left\|b\cdot\mathbf{V}^{-1}\tilde{\boldsymbol{x}}\right\|_p + b\gamma\cdot\left\|\mathbf{V}^{-1}\right\|_p,\end{aligned} \tag{12}$$

where (a) is due to the sub-multiplication property [Horn and Johnson, 2012], and hence

$$\left|\left\|b\cdot\mathbf{V}^{-1}\hat{\boldsymbol{x}}\right\|_p - \left\|b\cdot\mathbf{V}^{-1}\tilde{\boldsymbol{x}}\right\|_p\right| \le b\gamma\cdot\left\|\mathbf{V}^{-1}\right\|_p$$

For the second component on the right-hand side of the inequality in (11), letting $\hat{\boldsymbol{y}}=\mathbf{B}^{-1}\hat{\boldsymbol{x}},\ \tilde{\boldsymbol{y}}=\mathbf{B}^{-1}\tilde{\boldsymbol{x}}$, we can have

$$\left\|\hat{\boldsymbol{y}}-\tilde{\boldsymbol{y}}\right\|_p \le \left\|\mathbf{B}^{-1}\right\|_p\left\|\boldsymbol{\delta}\right\|_p \le \gamma\left\|\mathbf{B}^{-1}\right\|_p,$$

$$\begin{aligned}\left|\left\|\mathbf{T}\lceil\hat{\boldsymbol{y}}\rceil+\mathbf{C}\right\|_p - \left\|\mathbf{T}\widetilde{\boldsymbol{y}}+\mathbf{C}\right\|_p\right| &\overset{\text{(b)}}{\le} \left\|\mathbf{T}\lceil\hat{\boldsymbol{y}}\rceil+\mathbf{C}-(\mathbf{T}\widetilde{\boldsymbol{y}}+\mathbf{C})\right\|_p \\ &\le \left\|\mathbf{T}\right\|_p\left\|(\lceil\hat{\boldsymbol{y}}\rceil-\hat{\boldsymbol{y}})+(\hat{\boldsymbol{y}}-\tilde{\boldsymbol{y}})\right\|_p \\ &\le \left\|\mathbf{T}\right\|_p(\left\|\lceil\hat{\boldsymbol{y}}\rceil-\hat{\boldsymbol{y}}\right\|_p+\left\|\hat{\boldsymbol{y}}-\tilde{\boldsymbol{y}}\right\|_p) \\ &\le \left\|\mathbf{T}\right\|_p(N^{\frac{1}{p}}+\gamma\left\|\mathbf{B}^{-1}\right\|_p).\end{aligned} \tag{13}$$

where (b) has been proved in [Weisstein, 2023].
Combining equations (12) and (13) and letting $M_\gamma = b\gamma\cdot\left\|\mathbf{V}^{-1}\right\|_p + num\cdot\left\|\mathbf{T}\right\|_p(N^{\frac{1}{p}}+\gamma\left\|\mathbf{B}^{-1}\right\|_p)$, we have

$$\left|\Phi(\hat{\boldsymbol{x}})-\widetilde{\Phi}(\tilde{\boldsymbol{x}})\right| \le M_\gamma.$$

□

References

[Brown *et al.*, 2020] T Brown, B Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, and A Askell. Language models are few-shot learners advances. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, page 33, 2020.

[Chen *et al.*, 2022] Fahao Chen, Peng Li, Celimuge Wu, and Song Guo. Hare: Exploiting inter-job and intra-job parallelism of distributed machine learning on heterogeneous gpus. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, pages 253–264, 2022.

[Fan *et al.*, 2021] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021.

[Horn and Johnson, 2012] Roger A Horn and Charles R Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

[Jayaram Subramanya *et al.*, 2023] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 642–657, 2023.

[Li *et al.*, 2020] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[Li *et al.*, 2022] Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. Amp: Automatically finding model parallel strategies with heterogeneity awareness, 2022.

[Liu *et al.*, 2023] Ji Liu, Zhihua Wu, Danlei Feng, Minxu Zhang, Xinxuan Wu, Xuefeng Yao, Dianhai Yu, Yanjun Ma, Feng Zhao, and Dejing Dou. Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. *Future Generation Computer Systems*, 148:106–117, 2023.

[Mo *et al.*, 2024] Zizhao Mo, Huanle Xu, and Chengzhong Xu. Heet: Accelerating elastic training in heterogeneous deep learning clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 499–513, 2024.

[Narayanan *et al.*, 2019] 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.

[Rasley *et al.*, 2020] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong 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*, pages 3505–3506, 2020.

[Ren *et al.*, 2021] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564, 2021.

[Sergeev and Balso, 2018] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow, 2018.

[Shazeer *et al.*, 2018] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh-tensorflow: Deep learning for supercomputers, 2018.

[Shoeybi *et al.*, 2020] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

[Unger *et al.*, 2022] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, et al. Unity: Accelerating {DNN} training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 267–284, 2022.

[Verbraeken *et al.*, 2020] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33, 2020.

[Wang *et al.*, 2022] Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. Webformer: The web-page transformer for structure information extraction. In *Proceedings of the ACM Web Conference 2022*, pages 3124–3133, 2022.

[Weisstein, 2023] Eric W. Weisstein. Norm — mathworld, 2023. Accessed: 2023-10-05.

[Yang *et al.*, 2024] Fei Yang, Shuang Peng, Ning Sun, Fangyu Wang, Yuanyuan Wang, Fu Wu, Jiezhong Qiu, and Aimin Pan. Holmes: Towards distributed training across clusters with heterogeneous nic environment. In *Proceedings of the 53rd International Conference on Parallel Processing*, pages 514–523, 2024.

[Yi *et al.*, 2020] Xiaodong Yi, Shiwei Zhang, Ziyue Luo, Guoping Long, Lansong Diao, Chuan Wu, Zhen Zheng, Jun Yang, and Wei Lin. Optimizing distributed training deployment in heterogeneous gpu clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 93–107, 2020.

[Yu *et al.*, 2022] Menglu Yu, Ye Tian, Bo Ji, Chuan Wu, Hridesh Rajan, and Jia Liu. Gadget: Online resource optimization for scheduling ring-all-reduce learning jobs. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1569–1578. IEEE, 2022.

[Yuan *et al.*, 2022] Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Re, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 35:25464–25477, 2022.

[Zhao *et al.*, 2023] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023.