HETHUB: A DISTRIBUTED TRAINING SYSTEM WITH HETEROGENEOUS CLUSTER FOR LARGE-SCALE MODELS

Si Xu [1 *] **Zixiao Huang** [2 1 *] **Yan Zeng** [3 1 *] **Shengen Yan** [1] **Xuefei Ning** [2] **Quanlu Zhang** [1] **Haolin Ye** [1] **Sipei Gu** [1]
Chunsheng Shui [1] **Zhezheng Lin** [1] **Hao Zhang** [4] **Sheng Wang** [4] **Guohao Dai** [5 1 +] **Yu Wang** [2]

ABSTRACT

Training large-scale models relies on a vast number of computing resources. For example, training the GPT-4 model (1.8 trillion parameters) requires 25000 A100 GPUs . It is a challenge to build a large-scale cluster with one type of GPU-accelerator. Using multiple types of GPU-accelerators to construct a large-scale cluster is an effective way to solve the problem of insufficient homogeneous GPU-accelerators. However, the existing distributed training systems for large-scale models only support homogeneous GPU-accelerators, not support heterogeneous GPU-accelerators. To address the problem, this paper proposes a distributed training system with hybrid parallelism , HETHUB, for large-scale models, which supports heterogeneous cluster, including AMD, Nvidia GPU and other types of GPU-accelerators . It introduces a distributed unified communicator to realize the communication between heterogeneous GPU-accelerators, a distributed performance predictor and an automatic parallel planner to develop and train models efficiently with heterogeneous GPU-accelerators. Compared to the distributed training system with homogeneous GPU-accelerators, our system can support six combinations of heterogeneous GPU-accelerators. We train the Llama-140B model on a heterogeneous cluster with 768 GPU-accelerators(128 AMD and 640 GPU-accelerator A). The experiment results show that the optimal performance of our system in the heterogeneous cluster has achieved up to 97.49% of the theoretical upper bound performance.

1 INTRODUCTION

With the rapid development of artificial intelligence technology, large language models (LLMs) like GPT-4 (Achiam et al., 2023), Pangu (Zeng et al., 2021), M6 (Lin et al., 2021), and others (Brown et al., 2020; Du et al., 2021; Radford et al., 2019) have grown rapidly, and these models are widely used in various fields (Bi et al., 2023; Nori et al., 2023) due to their exceptional performance. However, their parameter scales are typically enormous, ranging from millions to billions, trillions, or even quadrillions. It needs a large number of GPU-accelerators to train these large-scale models. For example, GPT-4, with approximately 1.8 trillion parameters, requires around 25,000 A100 GPUs for its training.

At present, researchers mainly use Megatron-LM (Shoeybi et al., 2019), DeepSpeed (Rasley et al., 2020), or Pytorch with NVIDIA GPU, such as A100, V100, H800, to train large-scale models. In recent years, many GPU-accelerators from different manufacturers have developed rapidly. However, it is very hard to build a large-scale cluster with a single type of GPU-accelerator in many scenarios. Training the large-scale model with multiple types of GPU-accelerators is an effective way to solve the problem of insufficient homogeneous GPU-accelerator. For example, the Llama3-70B model with 70 billion parameters needs 900 Nvidia H100 GPUs to train over 10 months. If there are 500 Nvidia H100 GPUs (1000 TOPs), 400 Huawei GPU-accelerators(320 TOPs), and 700 AMD GPU-accelerators (383 TOPs), we can't train the Llama3-70B model with a single type of GPU-accelerators, as neither one type of GPU-accelerator can meet the demand. But if we use all types of GPU-accelerators, we can train the Llama3-70B model. Thus, building a heterogeneous cluster with different types of GPU-accelerators to train large-scale models, is a good way to solve the problem of large-scale computing resources.

The existing distributed training systems (Li et al., 2014; Mnih and Hinton, 2008; Luo et al., 2020) for large-scale models only support homogeneous cluster. It is challenging to train large-scale models with heterogeneous clusters due to the differences in architecture and software between different types of GPU-accelerators. **1) Communication challenge.** Different types of GPU-accelerators cannot communicate directly with each other, as different types of GPU-

*Equal contribution +Project lead [1]Infinigence-AI [2]Tsinghua University [3]Hangzhou Dianzi University [4]China Mobile Research Institute [5]Shanghai Jiao Tong University. Correspondence to: Guohao Dai <daiguohao@sjtu.edu.cn>, Yu Wang <yu-wang@tsinghua.edu.cn>.

accelerators have different communication libraries, such as Nvidia GPUs use NCCl, GPU-accelerator C use HCCL. **2) Development and training challenge.** It is very difficult to design and implement an optimal distributed training strategy for large-scale models in a heterogeneous cluster. The differences in computation and storage of different types of GPU-accelerators and the computation-communication strong coupling characteristic of large-scale models result in the exponential increase of the number of distributed strategies with the number of heterogeneous GPU-accelerators, layers or operators of models. **3) Accuracy challenge.** The accuracy difference of operators on different types of GPU-accelerators will make the accuracy of the model difficult to reach the accuracy of the homogeneous cluster.

This paper focuses on solving the communication development and training challenges, and proposes a distributed training system with heterogeneous clusters for large-scale models. Our major contributions can be summarized as follows:

1. We construct a distributed unified communicator to support communication between different GPU-accelerators. This communicator includes two communication libraries, one is a CPU-based communicator with Ethernet or IPoIB; the other one is a GPU-based communicator with IB or RoCE, which defines a unified communication interface to adapt to multiple types of GPU-accelerators.
2. A distributed performance predictor is proposed to help evaluating the distributed training strategy of models on the heterogeneous cluster. We conduct automatic profiling on a small cluster and build the performance evaluation model. Then, this performance evaluation model can be used to make performance predictions to guide the decision of the distributed training strategy on larger-scale clusters.
3. We introduce an automatic parallel planner, which can automatically search an optimal distributed parallelism strategy for the given model and heterogeneous cluster topology. It can enhance development and model computation efficiency.
4. We verify the performance and scalability of our system HETHUB with the Llama2-140B model in a heterogeneous cluster with 768 GPU-accelerators, which include 128 AMD GPU-accelerators and 640 GPU-accelerators A.

2 BACKGROUND

The success of large-scale models in natural language processing, machine translation, and other fields, stems from their vast number of parameters, which can fully mine the data characteristics. Distrinbuted training method is essential for large-scale models, which involves data parallelism, tensor parallelism, pipeline parallelism, and auto-parallel strategy. And if we use different types of GPU accelerators to train a large-scale model, it also involves heterogeneous training. Next, we will present these strategies in detail.

2.1 Data Parallelism

Data parallelism (DP) (Sergeev and Balso, 2018) is a parallelism method with data segmentation. In data parallelism, the dataset is split into multiple sub-datasets, while the model is replicated across GPU-accelerators. Each GPU-accelerator trains its assigned sub-dataset separately and updates model parameters after forward and backward propagation (Li et al., 2020). However, a challenge arises with memory redundancy as each GPU-accelerator stores duplicate copies of model parameters, optimizer states (Duchi et al., 2011), and gradients. To address this, DeepSpeed (Rasley et al., 2020) introduced the Zero Redundancy Optimizer (Rajbhandari et al., 2019), distributing model parameters, optimizer states, and gradients across devices. Consequently, during data updates, GPU-accelerators only need to update their respective partitioned parameters instead of the entire model.

2.2 Tensor Parallelism

Tensor parallelism (TP) (Shoeybi et al., 2019) involves simultaneously processing different parts of a tensor across devices. In tensor parallelism, input tensors are partitioned into subtensors and distributed across devices. After completing the computation of a certain layer, a device may transmit partial results to the next layer or other devices for further processing. Thus, communication operations like AllReduce and Broadcast are implemented typically between different layers or within a layer, to facilitate data exchange and synchronization between devices. However, since AllReduce (Sergeev and Balso, 2018) requires communication between all nodes and Broadcast needs to transmit data from one node to other nodes, the communication overhead of these two methods is significant with a large number of nodes.

2.3 Pipeline Parallelism

Pipeline parallelism (PP) segments the entire model into multiple-stages, and schedules these stages to different nodes or GPU-accelerators, where one stage includes 1 to k model layers. In pipeline parallelism, batches are typically divided into several micro-batches within each pipeline stage, and parameters for each stage are distributed to the respective computational GPU-accelerators during model initialization. This approach optimizes end-to-end training time by overlapping computation and communication across

different stages and micro-batches.

Current mainstream pipeline parallelism schemes include Gpipe (Huang et al., 2019), PipeDream-1F1B (Harlap et al., 2018), PipeDream-2BW (Narayanan et al., 2021), and Chimera (Li and Hoefler, 2021). Gpipe divides each stage's batch into multiple micro-batches that are computed sequentially, allowing the computation of micro-batches from different stages to overlap. However, in this arrangement, a stage must wait for the forward pass of all micro-batches to complete before commencing the backward pass, necessitating the storage of activations for the number of micro-batches, which results in significant memory consumption. Conversely, the PipeDream-1F1B arrangement interleaves forward and backward computations, storing the activation values of n pipeline stages at most, thereby reducing memory usage. Although PipeDream-1F1B has a similar pipeline utilization with Gpipe, it introduces a great optimization to the bubble ratio. PipeDream-2BW takes PipeDream-1F1B further by dividing the pipeline into two buckets to improve the overlap of computation and communication. It potentially leads to even higher throughput and better utilization, though at the cost of increased complexity and higher communication load, which are not supported in heterogeneous clusters. Chimera is a bi-directional scheduled pipeline method. Two micro batches start to train at the same time from the first and the last pipeline stage. So it has less pipeline bubble compared to the common one-directional pipeline. However, this method needs each stage to contain two parts of the model parameter, which adds a memory burden to the GPU-accelerator .

2.4 Auto-Parallel Strategy

Due to the complexity of model structures, the number of distributed training strategies for models increases exponentially with the number of model layers or operators. This makes manual tuning of distributed strategies based on expert experience highly demanding. To address this issue, researchers have proposed automatic parallelism methods. These methods utilize dynamic programming or graph algorithms to automatically search for distributed training strategies for models. These approaches can enhance efficiency by automatically identifying optimal or near-optimal distributed training strategies.

In recent years, automatic parallelism algorithms have developed rapidly(Fan et al., 2021; Santhanam et al., 2021; Schaarschmidt et al., 2021), such as FlexFlow (Jia et al., 2019), D-rec (Wang et al., 2021), Piper(Tarnawski et al., 2021), and Alpa (Zheng et al., 2022). FlexFlow constructs a SOAP search space and employs the Markov Chain Monte Carlo (MCMC) algorithm to identify the optimal intra-operator parallelism strategy. The Double Recursive Algorithm (D-rec) considers both intra-operator and inter-operator communication costs, utilizing a two-layer recursive algorithm to determine the optimal tensor sharding strategy. These methods can only search for partial parallelism strategies. For training large-scale models, it is essential to combine multiple parallelism methods. Piper introduces two-level dynamic programming to search for the distributed parallel strategy for data parallelism and model parallelism. Alpa supports the joint search of multiple parallelism strategies, divides the strategies into intra-operator parallelism and inter-operator parallelism, and optimizes intra-operator parallelism using integer linear programming while arranging inter-operator parallelism using dynamic programming. However, these methods only support training large-scale models under homogeneous clusters, not support or work well in heterogeneous clusters.

2.5 Heterogeneous Training

Unlike distributed training for large-scale models in a homogeneous cluster with one type of GPU-accelerators, it introduces communication and computational challenges in a heterogeneous cluster with multiple types of GPU-accelerators. For communication, it impedes the transfer of data between different types of GPU-accelerators, as different types of GPU-accelerators have their own communication libraries, which are not compatible with each other. For computation, existing distributed training strategies can lead to load imbalance, due to the balanced partitioning of tasks and the differences in computing resources of different types of GPU-accelerators.

Existing frameworks that support heterogeneous clusters include BPT-CNN (Chen et al., 2018), AccPar (Song et al., 2020), and Whale (Jia et al., 2022). BPT-CNN, a data parallelism algorithm for heterogeneous clusters, allocates setting the number of batches based on the computing resources of GPU-accelerators. It dynamically adjusts batch distribution after each training iteration according to the actual performance of GPU-acclerators. AccPar, a tensor parallelism algorithm, employs dynamic programming to derive optimal tensor-sharding strategies for each operator. These methods are limited to single parallelism strategy . Whale supports both data parallelism and pipeline parallelism . It employs data parallelism partition according to computing resources of GPU-accelerators and arranges pipeline parallelism stages with memory resources of GPU-accelerators in descending order. In conclusion, compared to training a model in a homogeneous cluster, it needs to solve two major problems in a heterogeneous cluster, the communication between different types of GPU-accelerators , and the efficiency of development and training caused by the complexity of model structure and the performance differences between multiple types of GPU-accelerators.

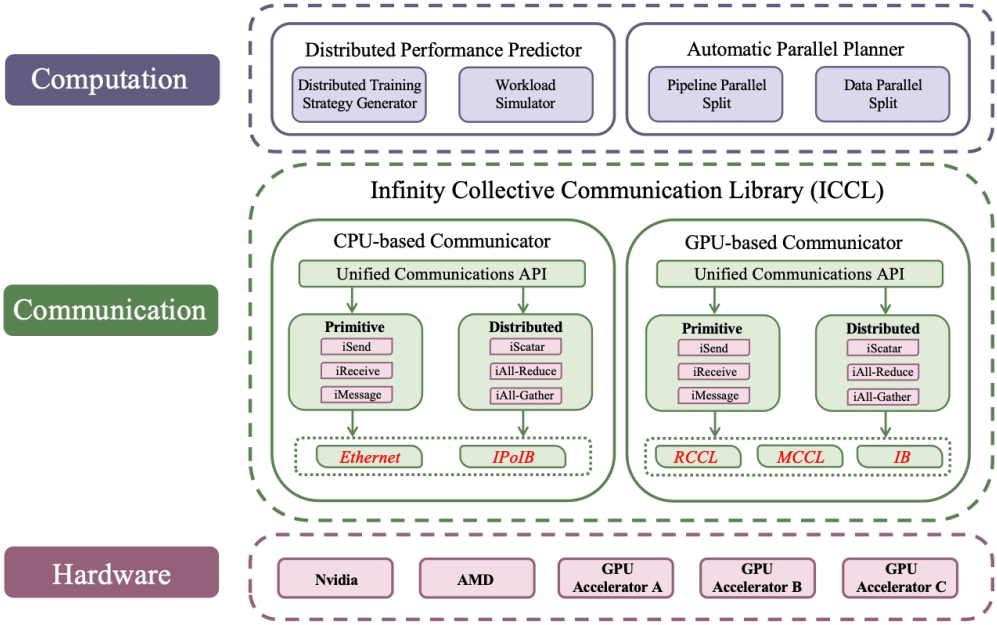

Figure 1: The scheme of HETHUB system

3 Design and Implementation

HETHUB, a distributed training system based on Megatron-LM and Megatron-DeepSpeed (Smith et al., 2022), for large-scale models in a cluster with different types of GPU-accelerators. We make optimization in the computing and communication level based on the differences of heterogeneous hardware, compared with the existing distributed training system in homogeneous clusters. As shown in Fig. 1, we design an infinity collective communication library(ICCL) based on Gloo (Facebook, 2023) to support the communication between heterogeneous GPU-accelerators. It includes a CPU-based communicator with Ethernet or IPoIB (Kashyap, 2006) and a GPU-based communicator with IB. At the communication level, we propose a distributed performance predictor to predict the training performance of large-scale models, and an automatic parallel planner to search for efficient distributed training strategies in heterogeneous clusters for large-scale models.

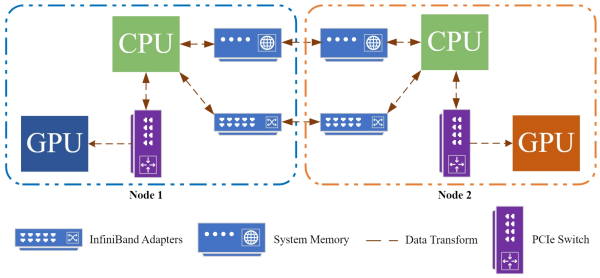

Figure 2: CPU-based communicator with Ethernet or IPoIB

3.1 Infinity Collective Communication Library

ICCL, including CPU-based communicator and GPU-based communicator, is proposed to solve the communication problem between different types of GPU-accelerators. It is implemented with Gloo, and supports two types of communication modes, CPU-based communication and GPU-based communication.

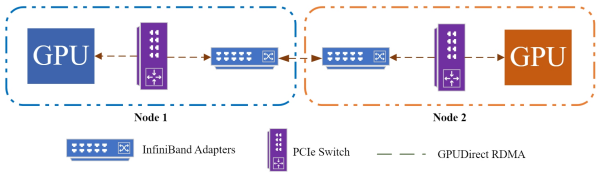

Figure 3: GPU-based communicator with IB

CPU-based Communicator. This communicator is designed for the scalability of heterogeneous clusters for different types of GPU-accelerator, it supports a new type of GPU-accelerator to join heterogeneous clusters quickly and at a low cost for training large-scale models. Physically, the CPU and the GPU-accelerator are connected through PCIE. CPUs and GPU-accelerators in a node directly communicate with each other via PCIE, and CPUs on different nodes communicate with each other with IPoIB or Ethernet. With the CPU-based communicator, the data, such as model parameters, needs to be copied from the GPU-accelerator to the CPU with PCIE, and then transferred to the CPU of target nodes with Ethernet or IPoIB, and lastly copied the

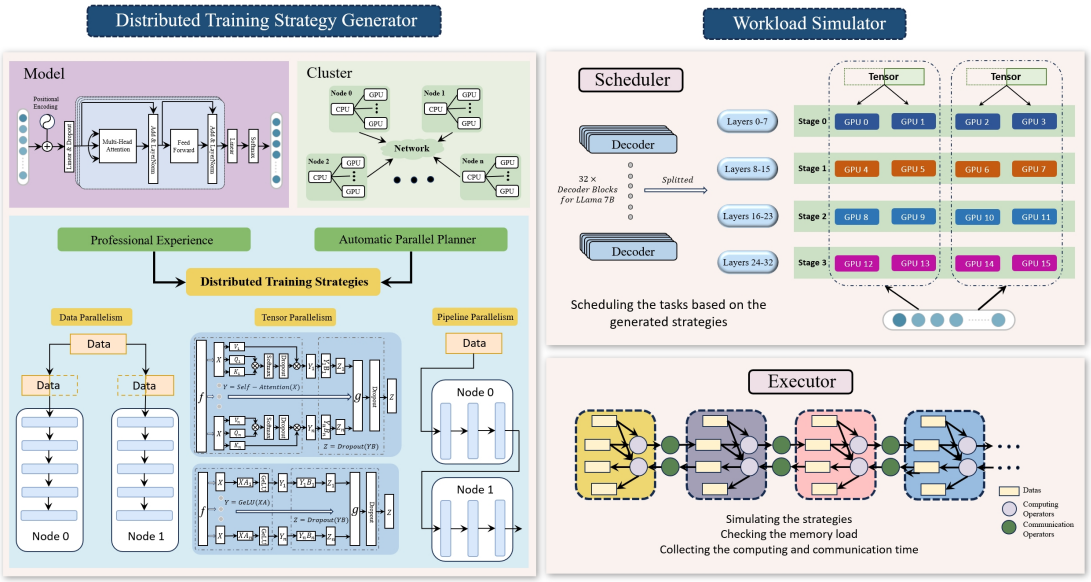

Figure 4: The workflow of distributed performance predictor

data from the CPU to the GPU-accelerators on the target nodes with PCIE, as shown in Fig. 2. This method avoids the differences in communication libraries of different GPU-accelerators by increasing the copy overhead between CPU and GPU-accelerators.

GPU-based Communication. This communicator is designed based on the RDMA (Kalia et al., 2014) for direct communication between heterogeneous GPU-accelerators via IB. Communication is a bottleneck problem in large-scale model training, because of the strong coupling between computation and communication of large-scale models. Different manufacturers have developed special communication libraries for GPU-accelerators to improve communication performance. However, the differences between communication libraries, such as APIs, data types, etc., cause different types of GPU-accelerators to not communicate directly with each other. We define a unified set of distributed communication protocols for different types of GPU-accelerators. It includes data type, communication primitive(e.g. iSend/iRecive), distributed communication function(e.g. iAllReduce, iAll-to-All, and etc). This method has high communication efficiency, but it needs different manufacturers of GPU-accelerators to adapt the unified communication protocol.

3.2 Distributed Performance Predictor

The distributed performance predictor is designed to find distributed training strategies for large-scale models in heterogeneous clusters with low cost. It proposes a distributed training strategy generator to construct different strategies for large-scale models, and presents a workload simulator to simulate and train models with generated distributed training strategy. The optimal distributed training strategy searched by the distributed performance predictor will be used to train large-scale models in real heterogeneous clusters.

Distrbiuted Training Strategy Generator. It constructs a computed graph with weights to describe the structure of large-scale models, where the weights represent the computation performance of the operator or layer of transformer models, sampled from different types of GPU-accelerators. It also samples the computing(e.g. TFLOPs in fp16 precision) and storage information of different GPU-accelerators, as well as the communication bandwidth between heterogeneous GPU-accelerators. Based on this, we use the expert experience or automatic parallel method to generate distributed training strategies for large-scale models, including pipeline parallelism, data parallelism, and tensor parallelism.

比较粗粒度的估计，枚举很多种可行的 parallelism plan

Workload Simulator. The workload simulator is developed to simulate the execution of model training tasks in real heterogeneous clusters. It will calculate the overhead of training the large-scale model, such as the time cost for each iteration step and max memory usage, according to the distributed training strategy generated by the distributed training strategy generator and the operator's execution performance on different types of GPU-accelerators. Lastly, it will give the optimal distributed parallel strategy for a large-scale model from multiple strategies generated by the distributed training strategy generator, to guide the deployment and training of the model in a real heterogeneous cluster.

类似 alpa 那样对每个 OP 做 profile

前面强调了在 small cluster 上运行，应该是对每个 stage 串行地在该 small cluster 上独立跑，而不是进行 OP 级别的拆分

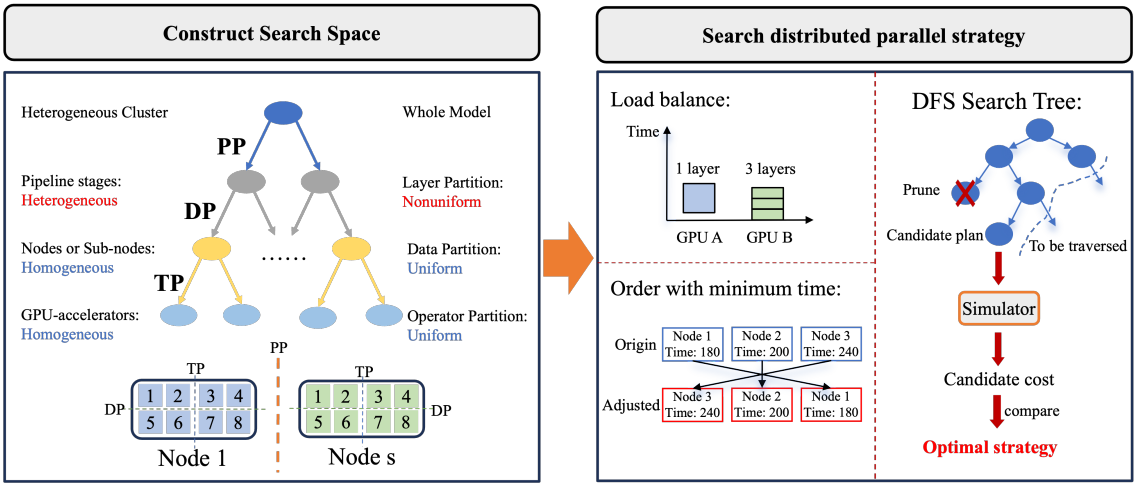

Figure 5: The workflow of automatic parallel planner

3.3 Automatic Parallel Planner

It is an NP-hard problem to find an optimal distributed training strategy for a large-scale model(Kennedy and Kremer, 1998; Liang et al., 2023), as the number of distributed training strategies increases exponentially with the increase of the number of model layers or operators, especially in heterogeneous clusters. The goal of the automatic parallel planner is to find an optimal distributed training strategy for the model automatically. It automatically divides the large-scale model into multiple sub-models and schedules them to different GPU-accelerators, according to the execution performance of the model layer or operator on different GPU-accelerators. It supports data parallelism, tensor parallelism, and pipeline parallelism for models.

Considering the communication performance between heterogeneous GPU-accelerators is lower than that between homogeneous GPU-accelerators, we employ data parallelism combined with intra-node tensor parallelism on homogeneous nodes and pipeline parallelism across heterogeneous nodes. To ensure the correctness of training results, pipeline parallelism satisfies the data constraints of the Pipedream-1F1B scheme. The specific design is illustrated in the Fig. 5.

Construct search space. A three-level search tree is constructed to represent the search space of distributed training strategies for models. Where the root node represents the whole model, and the other nodes represent the sub-models after splitting. Furthermore, the leaf nodes represent the final models that are executed in a single GPU-accelerator. The first layer uses a non-uniform pipeline parallelism splitting strategy to split the model based on the total number of transformer layers. The purpose of the splitting is to ensure load balancing of the computation for different types of GPU-accelerator. The second layer splits the sub-model into homogeneous nodes using a uniform data parallelism strategy. The third layer splits the model into GPU-accelerators using a uniform tensor parallelism strategy. After three layers of splitting, the complete model can be mapped to a heterogeneous cluster for training.

Search distributed training strategy. To make full use of heterogeneous GPU-accelerator resources, we give two rules with the goal of load balancing and minimum end-to-end training time, to guide distributed parallelism strategy searching in the constructed search tree.

1) Load balance. According to the computing resources of the heterogeneous GPU-accelerator and the computing requirements of the model layer, we divide the model layer irregularly to balance the computing tasks among different GPU-accelerators as much as possible. That is, GPU-accelerators with high computing resources perform more layers.

2) Minimum end-to-end training time. We schedule stages in pipeline parallelism to different types of GPU-accelerators to optimize the end-to-end training time, according to the execution time of stages on different types of GPU-accelerators and the communication time between the stages.

Based on the above rules, we find a distributed training strategy by using the DFS algorithm to traverse the search tree, and use the distributed performance predictor to evaluate the training time of the model with the searched distributed training strategy. Ultimately, we select the distributed training strategy with the lowest evaluation cost.

4 EXPERIMENTS

In order to verify the effectiveness and performance of the HETHUB system, this paper uses the models of Llama2 to carry out heterogeneous training experiments on NV, AMD, and other types of GPU-accelerators. In addition, this paper compares the throughput and MFU performance with the homogeneous cluster environment, and also analyses the end-to-end time of model training on a heterogeneous cluster with 768 GPU-accelerators.

4.1 Experimental Setup

Homogeneous clusters. We use 2 representative homogeneous clusters, including the AMD GPU-accelerator and GPU-accelerator A clusters. The AMD cluster includes 20 worker nodes with 160 GPU-accelerators, each worker node has 8 GPU-accelerators connected to 192 CPU cores via PCIe switches. The GPU-accelerator A cluster includes 96 worker nodes with 768 GPU-accelerators, each worker node has 8 GPU-accelerators connected to 128 CPU cores via PCIe switches. And infiniband (IB) is used for homogeneous clusters, where the bandwidth is 200 Gb/s.

Heterogeneous cluster. We use 4 heterogeneous clusters with the ratio of AMD and GPU A being 1:5, including a 12-nodes cluster with 96 GPU-acclerators(12N96D), a 24-nodes cluster with 192 GPU-accelerators(24N192D), a 48-nodes cluster with 384 GPU-accleraotrs(48N384D) and a 96-nodes cluster with 768 GPU-accelerators(96N768D). Infiniband (IB) is used for homogeneous nodes, where the bandwidth is 200 Gb/s. Ethernet is used for heterogeneous nodes, where bandwidth is 25 Gb/s.

Experiment parameters. We train Llma2 models with the hybrid distributed parallelism strategy with pipeline parallelism, data parallelism, and tensor parallelism. For the heterogeneous cluster, we employ data parallelism combined with intra-node tensor parallelism on homogeneous nodes and pipeline parallelism across heterogeneous nodes. The details of the configurations are shown in Table 1. Where DP represents data parallelism, TP represents tensor parallelism, PP represents pipeline parallelism, NUM is the number of nodes, global-batch-size=2048*NUM/10 for homogeneous clusters, and global-batch-size=2048*NUM/6 for heterogeneous clusters, DP=NUM*8/TP/PP and TP=1 for all clusters, and the ratio of heterogeneous clusters is AMD:GPU A = 1:5.

4.2 Model and Dataset

Model. We choose the language model Llama2 as the test model. The Llama2 (Touvron et al., 2023) is a collection of pre-trained and fine-tuned large language models (LLMs) ranging in scale from 7 billion to 140 billion parameters, including Llama2-7B, Llama2-13B, Llama2-35B, Llama2-70B and Llama2-140B.

DataSet. We conduct experiments on the Dolma and MAP-CC datasets. Dolma Dataset (Soldaini et al., 2024) is an open dataset of 3 trillion tokens from a diverse mix of web content, academic publications, code, books, and encyclopedic materials. MAP-CC (Massive Appropriate Pretraining Chinese Corpus) dataset (Du et al., 2024) is a large-scale open-source Chinese pre-training dataset developed by Multimodal Art Projection, Fudan University, Peking University, and other organizations. The dataset contains 80 billion tokens and consists of multiple subsets, each from different data sources, such as blogs, news articles, Chinese encyclopedias, Chinese academic papers, and Chinese books.

4.3 Metrics

Throughput. Throughput is measured by the number of tokens calculated by one GPU-accelerator in one second. It can be calculated according to Eq. 1

$$TGS = \frac{L \times G}{S \times T} \tag{1}$$

L represents the sequence length of the training data, G means the global batch size of one iteration, S represents the number of GPU accelerators being used in the training process, and T represents the time spent on training one iteration.

MFU. Model FLOPs Utilization (MFU) is the ratio of the actual throughput to the theoretical maximum throughput assuming 100% of peak FLOPs.The definition of MFU is shown in Eq. 2. We calculate the MFU in the same environment as the previous section was used.

$$\text{MFU} = \frac{T_{test}}{T_{peak}} \times 100\% \tag{2}$$

where T_{test} stands for the tested TFLOPS in fp16 precision per GPU for cluster and T_{peak} refers to the theoretical peak TFLOPS in fp16 precision. For the heterogeneous cluster, the peak TFLOPS is the average value of all GPU-accelerators.

Communication. Communication is used to measure the cost of communication between different nodes for training models in this paper. The point-to-point communication T_{com} in the model training is defined as Eq.3:

$$T_{com} = B \times L \times H \times 2 \tag{3}$$

Where the B represents batch size, L is the sequence length and H denotes the number of features in each hidden state.

Table 1: Configuration of model training in homogeneous and heterogeneous clusters.

model	Layers	Hidden Size	PP	NUM
Llama2-7B	32	4096	12	12 24 48 96

(a) Configurations of Llama2-7B in heterogeneous clusters

model	Layers	Hidden Size	PP	NUM
Llama2-13B	40	5120	12	12 24 48 96

(b) Configurations of Llama2-13B in heterogeneous clusters

model	Layers	Hidden Size	PP	NUM
Llama2-35B	40	8192	12	12 24 48 96

(c) Configurations of Llama2-35B in heterogeneous clusters

model	Layers	Hidden Size	PP	NUM
Llama2-70B	80	8192	12	12 24 48 96

(d) Configurations of Llama2-70B in heterogeneous clusters

model	Layers	Hidden Size	PP	NUM
Llama2-140B	160	8192	24	12 24 48 96

(e) Configurations of Llama2-140B in heterogeneous clusters

model	Layers	Hidden Size	PP	cluster	NUM
Llama2-70B	80	8192	10	AMD	10 20
				GPU A	60 80 96

(f) Configurations of Llama2-70B in homogeneous clusters

4.4 Experiment Results

4.4.1 Throughput

We first use the Llama2-7B model to verify the effect of uniform and un-uniform stage segmentation of pipeline parallelism on the performance of a small heterogeneous cluster with 1 AMD GPU-accelerators and 5 GPU-accelerator A. Then, we train Llam2-7B, Llama2-13B, Llama2-35B, Llama2-70B and Llama2-140B on different heterogeneous clusters with un-uniform stage segmentation of pipeline parallelism, to evaluate the throughput of HETHUB. The experimental results show that HETHUB has good throughput performance.

The results in Fig.6 a) show that the performance of non-uniform stage segmentation is better than that of uniform stage segmentation in heterogeneous clusters. Especially, the un-uniform segmentation with PP = 12 has the highest throughput performance of 920.84 tokens/GPU-accelerators/s, it can improve by up to 2.5%, compared to the uniform segmentation of GPU-accelerator A clusters with PP=6. The experiment results from Fig.6 b)-f) show that the throughput of HETHUB remains stable with the increase of model parameter size and heterogeneous cluster size, and the throughput of HETHUB in heterogeneous clusters reaches 54.71% of an AMD cluster with 160 GPU-accelerators and 100.96% for a GPU-accelerator A cluster with 768 GPU-accelerators. For Llama2-70B, the throughput of AMD is 93.81 TFLOPs/GPU-accelerators, and the throughput of GPU-accelerator A is 48.08 TFLOPs/GPU-accelerators. In the heterogeneous cluster of AMD and More, the throughput reaches a maximum of 51.11 TFLOPs/GPU-accelerators, achieving 91.75% of the theoretical value for the heterogeneous cluster. However, if the number of cluster nodes increases without changing the model parameter size, the throughput decreases, as the computation per node decreases but cross-node communication increases.

4.4.2 MFU

To evaluate the scalability and computing performance of the HETHUB system further, we tested the MFU of training Llama2-70B model on the homogeneous cluster(Nvidia, AMD, GPU-accelerator A, GPU-accelerator B, and GPU-accelerator C) and heterogeneous cluster(including a 2-node

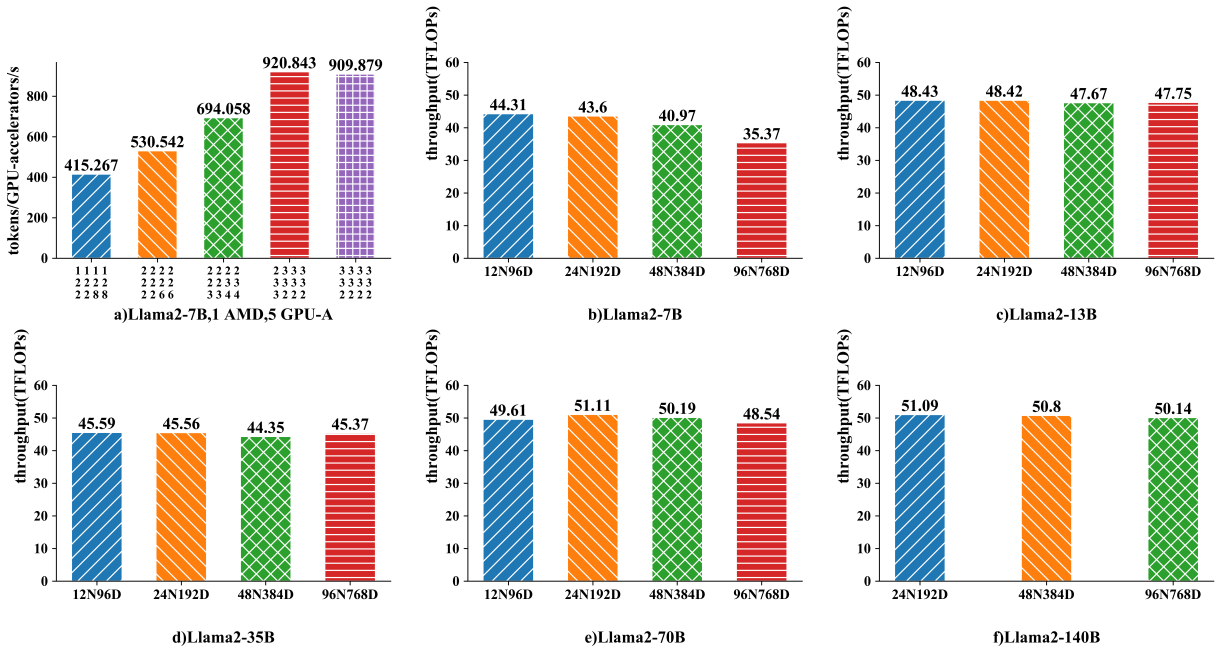

Figure 6: Throughput performance of HETHUB with different models in heterogeneous clusters

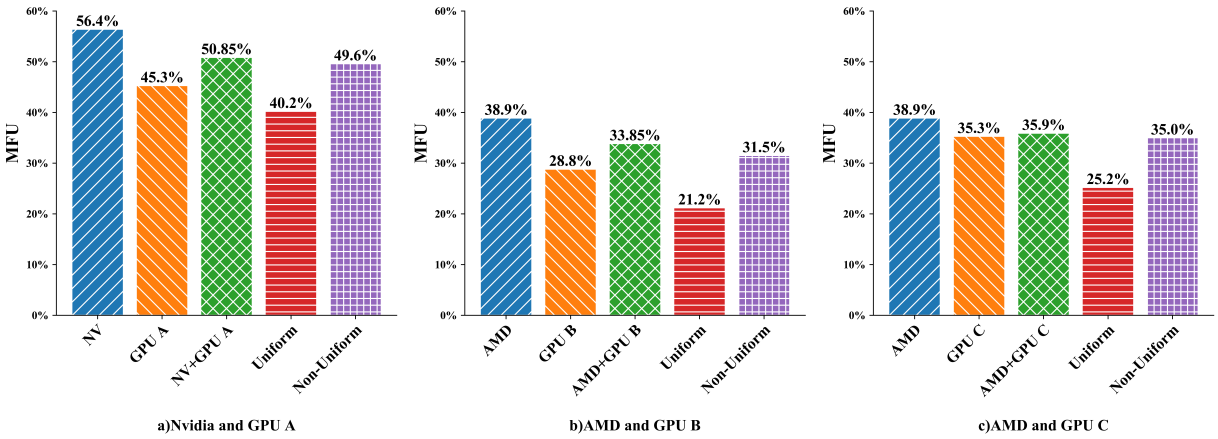

Figure 7: MFU for training Llama2-70B with uniform and non-uniform stage segmentation in pipeline parallism

cluster with Nvidia and GPU-accelerator A, a 2-node cluster with AMD and GPU-accelerator B, and a 120-node cluster with AMD and GPU-accelerator C). The experimental results show that the MFU of HETHUB can reach 97.49% of the theoretical MFU, compared to existing pipeline parallel methods with uniform stage segmentation. Where the PP = 10, a stage has 8 layers for the uniform segmentation method, and PP = 12, stages is 7 6 6 6 6 7 7 7 7 7 7 7, for the Non-uniform stage segmentation.

Specifically, in Fig.7 a), the MFU of Nvidia GPU-accelerator is 56.4%, the MFU of GPU-accelerator A is 45.3%, the MFU of heterogeneous clusters with Nvidia and GPU-accelerator A with non-uniform segmentation method is 49.60%, which can reach 97.54% of the theoretical MFU 50.85%. The MFU with the non-uniform segmentation method improves by 9.4% over the uniform segmentation method. The Fig.7 b), the MFU of AMD GPU-accelerator is 38.90%, the MFU of GPU-accelerator B is 28.80%, the MFU of heterogeneous clusters with AMD and GPU-accelerator B with non-uniform segmentation method is 31.50%, which can reach 93.05% of the the-

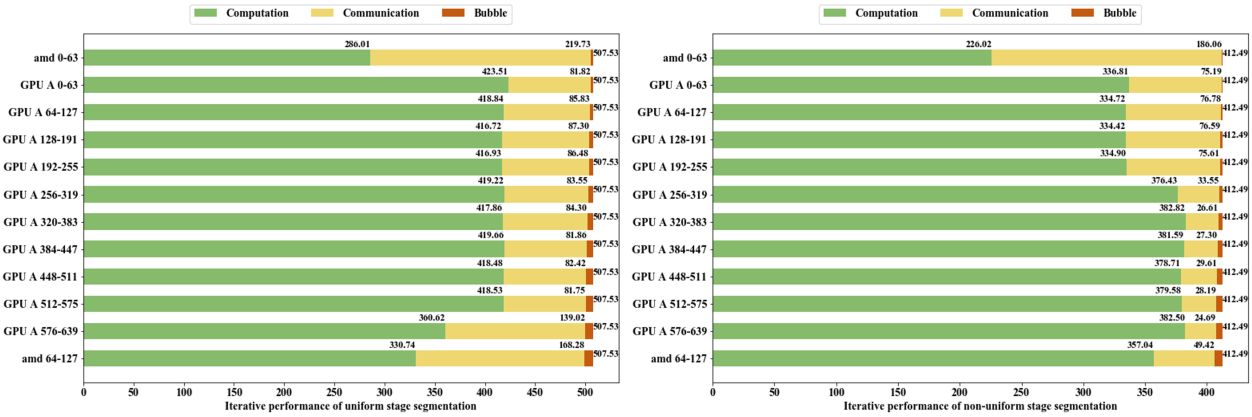

Figure 8: Iterative performance of non-uniform and uniform stage segmentation in pipeline parallism

oretical MFU 33.85%. The MFU with the non-uniform segmentation method improves by 10.3% over the uniform segmentation method. The Fig.7 c), the MFU of AMD GPU-accelerator is 38.90%, the MFU of GPU-accelerator C is 35.30%, the MFU of heterogeneous clusters with AMD and GPU-accelerator C with non-uniform segmentation method is 35.00%, which can reach 97.49% of the theoretical MFU 35.90%. The MFU with the non-uniform segmentation method improves by 9.8% over the uniform segmentation method.

From the above analysis, we can see that the HETHUB system has excellent scalability, it can support multiple types of GPU-accleartors for hybrid training large models. Moreover, the non-uniform segmentation method of pipeline parallelism is more suitable for heterogeneous clusters than the uniform segmentation method.

4.4.3 End-to-End Performance

We analyzed the end-to-end performance of the HETHUB system for training the Llama2-70B model on a heterogeneous cluster with 128 AMD GPU-accelerators and 640 GPU-accelerators A, using uniform and non-uniform stage segmentation method of pipeline parallelism. We use pipeline parallelism for models across heterogeneous nodes, AMD, and More. The theoretical bandwidth of Ethernet between the AMD node and the GPU-accelerator A node is 25 Gb/s, the actual bandwidth is 18 - 20 Gb/s in a real environment. The theoretical bandwidth of IB between the AMD node and GPU-accelerator A node is 200 Gb/s, the actual bandwidth is 160 - 180 Gb/s in the real heterogeneous cluster.

The results in Fig.8 show that the end-to-end performance of the non-uniform stage segmentation method of pipeline parallelism is higher than the uniform stage segmentation method, as the differences in the computing resources of different types of GPU-accelerators. The end-to-end time with non-uniform stage segmentation method of pipeline parallelism is 412.49ms, it improves by 18.69%, compared with the uniform stage segmentation method which needs 507.3ms.

5 Conclusion

In this work, we designed and implemented a distributed training system with hybrid parallelism, HETHUB, for training large-scale models in heterogeneous clusters, including Nvidia, and other types of GPU-accelerators. It supports the communication between different types of GPU-accelerators, and realizes the efficient development, deployment, and training of models through the automatic parallel planner and distributed performance predictor. Experiments demonstrate that the optimal performance of our system in the heterogeneous cluster has achieved up to 97.49% of the theoretical upper bound performance.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, et al. Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation. *arXiv preprint arXiv:2104.12369*, 2021.

Junyang Lin, Rui Men, An Yang, Chang Zhou, Ming Ding, Yichang Zhang, Peng Wang, Ang Wang, Le Jiang, Xianyan Jia, et al. M6: A chinese multimodal pretrainer. *arXiv preprint arXiv:2103.00823*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. All nlp tasks are generation tasks: A general pretraining framework, 2021.

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

Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619 (7970):533–538, 2023.

Harsha Nori, Nicholas King, Scott Mayer McKinney, Dean Carignan, and Eric Horvitz. Capabilities of gpt-4 on medical challenge problems. *arXiv preprint arXiv:2303.13375*, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*, pages 583–598, 2014.

Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. *Advances in neural information processing systems*, 21, 2008.

Qinyi Luo, Jiaao He, Youwei Zhuo, and Xuehai Qian. Prague: High-performance heterogeneity-aware asynchronous decentralized training. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 401–416, 2020.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018. URL http://arxiv.org/abs/1802.05799.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. 2020.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. pages 257–269, 2011.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimization towards training A trillion parameter models. *CoRR*, abs/1910.02054, 2019. URL http://arxiv.org/abs/1910.02054.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, pages 7937–7947. PMLR, 2021.

Shigang Li and Torsten 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*, pages 1–14, 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.

Keshav Santhanam, Siddharth Krishna, Ryota Tomioka, Andrew Fitzgibbon, and Tim Harris. Distir: An intermediate representation for optimizing distributed neural networks. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 15–23, 2021.

Michael Schaarschmidt, Dominik Grewe, Dimitrios Vytiniotis, Adam Paszke, Georg Stefan Schmid, Tamara Norman, James Molloy, Jonathan Godwin, Norman Alexander Rink, Vinod Nair, et al. Automap: Towards ergonomic automated parallelism for ml models. *arXiv preprint arXiv:2112.02958*, 2021.

Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

Haoran Wang, Chong Li, Thibaut Tachon, Hongxing Wang, Sheng Yang, Sébastien Limet, and Sophie Robert. Efficient and systematic partitioning of large and deep neural networks for parallelization. In *Euro-Par 2021: Parallel Processing: 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1–3, 2021, Proceedings 27*, pages 201–216. Springer, 2021.

Jakub M Tarnawski, Deepak Narayanan, and Amar Phanishayee. Piper: Multidimensional planner for dnn parallelization. *Advances in Neural Information Processing Systems*, 34:24829–24840, 2021.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022.

Jianguo Chen, Kenli Li, Kashif Bilal, Keqin Li, S Yu Philip, et al. A bi-layered parallel training architecture for large-scale convolutional neural networks. *IEEE transactions on parallel and distributed systems*, 30(5):965–976, 2018.

Linghao Song, Fan Chen, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. Accpar: Tensor partitioning for heterogeneous deep learning accelerators. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 342–355. IEEE, 2020.

Xianyan Jia, Le Jiang, Ang Wang, Wencong Xiao, Ziji Shi, Jie Zhang, Xinyuan Li, Langshi Chen, Yong Li, Zhen Zheng, et al. Whale: Efficient giant model training over heterogeneous {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 673–688, 2022.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Gloo 2023. Collective Communications Library with Various Primitives for Multi-Machine Training. *https://github.com/facebookincubator/gloo*, 2023

Vivek Kashyap. Ip over infiniband (ipoib) architecture. RFC 4392, April 2006. URL https://www.rfc-editor.org/info/rfc4392.

Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using rdma efficiently for key-value services. *ACM SIGCOMM Computer Communication Review*, 44(4):295–306, 2014.

Ken Kennedy and Ulrich Kremer. Automatic data layout for distributed-memory machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 20(4): 869–916, 1998.

Peng Liang, Yu Tang, Xiaoda Zhang, Youhui Bai, Teng Su, Zhiquan Lai, Linbo Qiao, and Dongsheng Li. A survey on auto-parallelism of large-scale deep learning training. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research, 2024.

Xinrun Du, Zhouliang Yu, Songyang Gao, Ding Pan, Yuyang Cheng, Ziyang Ma, Ruibin Yuan, Xingwei Qu, Jiaheng Liu, Tianyu Zheng, Xinchen Luo, Guorui Zhou, Binhang Yuan, Wenhu Chen, Jie Fu, and Ge Zhang. Chinese tiny llm: Pretraining a chinese-centric large language model, 2024.

Rui Zhang, Zhendong Yin, Zhilu Wu, and Siyang Zhou. A novel automatic modulation classification method using attention mechanism and hybrid parallel neural network. *Applied Sciences*, 11(3):1327, 2021.