Mobius: Fine Tuning Large-Scale Models on Commodity GPU Servers

Yangyang Feng
fyy21@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Minhui Xie
xmh19@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Zijie Tian
tzj21@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Shuo Wang
shvowang@tsinghua.edu.cn
Tsinghua University
Beijing, China

Youyou Lu
luyouyou@tsinghua.edu.cn
Tsinghua University
Beijing, China

Jiwu Shu*
shujw@tsinghua.edu.cn
Tsinghua University
Beijing, China

ABSTRACT

Fine-tuning on cheap commodity GPU servers makes large-scale deep learning models benefit more people. However, the low inter-GPU communication bandwidth and pressing communication contention on the commodity GPU server obstruct training efficiency.

In this paper, we present Mobius, a communication-efficient system for fine tuning large-scale models on commodity GPU servers.

The key idea is a novel pipeline parallelism scheme enabling heterogeneous memory for large-scale model training, while bringing fewer communications than existing systems. Mobius partitions the model into stages and carefully schedules them between GPU memory and DRAM to overlap communication with computation. It formulates pipeline execution into a mixed-integer program problem to find the optimal pipeline partition. It also features a new stage-to-GPU mapping method termed cross mapping, to minimize communication contention.

Experiments on various scale models and GPU topologies show that Mobius significantly reduces the training time by 3.8-5.1× compared with the prior art.

CCS CONCEPTS

• **Computer systems organization** → **Pipeline computing**; • **Computing methodologies** → *Neural networks*.

KEYWORDS

Neural networks, parallel training, distributed training

ACM Reference Format:
Yangyang Feng, Minhui Xie, Zijie Tian, Shuo Wang, Youyou Lu, and Jiwu Shu. 2023. Mobius: Fine Tuning Large-Scale Models on Commodity GPU Servers. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3575693.3575703

*Jiwu Shu is the corresponding author (shujw@tsinghua.edu.cn).

ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9916-6/23/03.
https://doi.org/10.1145/3575693.3575703

1 INTRODUCTION

Recent years have witnessed the great success of large-scale deep learning models. They deliver significant accuracy improvement in the fields of natural language processing (NLP) [16, 41] and computer vision (CV) [18, 21]. From Megatron (2018, [40]) to Meta-OPT (2022, [46]), the size of large-scale models has increased by over 200×, and this trend will continue [25]. Training such large-scale models from scratch requires enormous computing power, which is only feasible on supercomputers or datacenters of top-tier technology companies.

Fortunately, a pre-trained large-scale model can be reused for multiple different downstream tasks via *fine-tuning*. It only requires a short training (several GPU-days) on an existing pre-trained model with new data. Considering the costs of training from scratch (up to 4.5 million dollars for training GPT-3 [13]), for most machine learning practitioners, fine-tuning is the only practical way to enjoy the benefits of large-scale models.

In this paper, we consider the problem of fine-tuning large-scale models on low-cost *commodity GPU servers*. We focus on commodity GPUs (e.g., 3090-Ti [7]) since they provide similar computing performance to data center GPUs (e.g., A100 [11]), but at a 7× lower price. Thus, they are more accessible to most people. Although existing fine-tuning systems such as Microsoft DeepSpeed [3] support large-scale model training on a single DGX-2 server [4] by using heterogeneous memory, they are designed for data center GPUs. We find the existing system's communication pattern is mismatched with the ***scarce communication resources*** on commodity GPU servers, leading to a serious communication bottleneck (about 70% of training time using DeepSpeed is spent on communication in our evaluation).

First, the inter-GPU communication bandwidth on commodity GPU servers is low. Unlike data center GPUs equipped with NVLink, which delivers a bandwidth of up to 900 GB/s [12], commodity GPUs can only communicate with other GPUs using PCIe-3.0 with a bandwidth of 16 GB/s. However, existing works are based on ZeRO data parallelism [35], which stores model states distributedly in multiple GPUs to reduce redundancy. It needs frequent collective communication to transfer model states between GPUs, generating communication of 7.3× the model size in one training step, according to our analysis. Such a high communication traffic at a low inter-GPU bandwidth brings a lot of overhead.

Second, the inter-GPU communication contention is severe on commodity GPU servers. Since commodity GPUs lack GPUDirect peer to peer (GPUDirect P2P) [9] support, inter-GPU communication is first routed through CPU to DRAM and then transferred to the target GPU. Thus, when multiple GPUs transfer data simultaneously, there is serious bandwidth contention at CPU's root complexes. Unfortunately, existing works rely on massive all-to-all collective communications among all GPUs to transfer parameters and gradients, which aggravate contention and limit the available bandwidth per GPU further.

To this end, we propose Mobius, a communication-efficient system for large-scale model fine-tuning on commodity GPU servers. Mobius leverages a key observation: traditional pipeline parallelism [24, 31] is more suited for commodity GPUs than the existing system's ZeRO data parallelism [35–37], since it only transfers small activations and activation gradients between adjacent GPUs, bringing remarkably fewer communications. Different from existing pipeline parallelism systems that only support models that fit in GPU memory, Mobius pipeline enables large-scale model training by introducing heterogeneous memory. However, it, meanwhile, brings extra communication traffic. To reduce that, Mobius redesigns the pipeline scheme. Specifically, Mobius pipeline divides a model into *stages*, each of which contains several model layers. These stages are stored in DRAM. Each GPU is responsible for multiple stages' execution and alternates stages by swapping them between GPU and DRAM. To further reduce communication overhead, Mobius overlaps communication by prefetching the next stage.

Enhancing pipeline parallelism with heterogeneous memory requires Mobius to revisit two classic problems which determine the performance of pipelining training: 1) how to partition the model into stages, and 2) how to map each stage to GPUs.

First is how to partition the model into stages. Peaking efficiency of Mobius pipeline requires a load and communication balanced partition of the model, as the slowest stage in the pipeline limits the overall throughput. In Mobius pipeline, too small a stage increases communication overhead, due to more transfer of activations and activation gradients, while too large a stage causes not enough GPU memory for prefetching, losing opportunities to overlap computation and communication. Existing partition algorithms [24, 45] do not work since they are formulated in the all-in-GPU-memory scenario without considering prefetching and multi-stages. Mobius formulates pipeline execution as a mixed-integer program (MIP) problem and then finds an **optimal** partition scheme based on our model partition algorithm (called *MIP partition algorithm*). We also adopt *layer similarity* to reduce the profiling time when determining the parameters of MIP.

After partitioning, the second problem is how to map each stage to GPUs. We find that the naive sequential mapping scheme of existing pipeline systems [24, 31] is not PCIe topology-aware, thus can cause severe communication contention on the CPU root complexes of commodity GPU servers. To alleviate this contention, our key idea is to prevent GPUs under the same root complex from transferring data simultaneously, where possible. With this key idea, Mobius tries the best to map two adjacent stages to two GPUs under different CPU root complexes; we call this *cross mapping* scheme. Note that there may be multiple cross mapping schemes. Mobius automatically searches for the best one by estimating the communication contention degree based on the PCIe topology of the GPU server.

Table 1: Performance and price comparison of a 3090-Ti GPU and an A100 GPU. GPUDirect P2P enables GPU-to-GPU communications directly over the memory fabric (PCIe, NVLink). High-bandwidth connectivity means all GPUs in a node can be fully connected via NVLink and NVLink Switch.

	3090-Ti	A100
Price	$2,000	$14,000
FP32 Performance	40 TFlops	19 TFlops
Tensor Cores	336	432
GPUDirect P2P	not support	support
High-bandwidth Connectivity	not support	support

We evaluate Mobius with extensive experimental settings: different scales of models, batch sizes, GPU topologies, and GPUs. Compared with the state-of-the-art system DeepSpeed, Mobius significantly reduces the training time by 3.8 – 5.1×.

We make the following contributions in this paper:

- We profile existing fine-tuning systems on commodity GPU servers, and find that the scarcity of communication resources is the key performance bottleneck.
- We introduce a communication-efficient system, Mobius, for fine-tuning large-scale models on commodity GPU servers.
- We evaluate Mobius and show its efficiency.

2 BACKGROUND AND MOTIVATION

In this section, we identify the importance of fine tuning large-scale models (§2.1), and describe the opportunities and gaps of commodity GPUs (§2.2). Then we demonstrate existing work's limitations on commodity GPUs, by analyzing and profiling DeepSpeed (§2.3).

2.1 Fine Tuning Pre-trained Models

Fine tuning pre-trained models democratizes the benefits of large-scale models. Training a large model from scratch requires tremendous time (several days on thousands of GPUs [33]) and millions of money [38]. For example, training GPT-3 of 175B parameters requires 355 GPU-years, which costs over 4 million dollars [13]. Such a high cost makes it only affordable for a small proportion of people in the AI community. Fortunately, some organizations make their pre-trained models publicly available, such as Meta's Open Pre-trained Transformer (OPT) [40]. These pre-trained models can be fine tuned for different downstream tasks, which is proven to be effective [22]. Fine tuning requires less training time and cost compared to training a model from scratch. For most users, fine tuning these publicly available pre-trained large-scale models is only a practical way to enjoy their benefits.

2.2 Commodity GPU Server

From the perspective of computational power, commodity GPU server are a cheaper and more affordable choice for most people

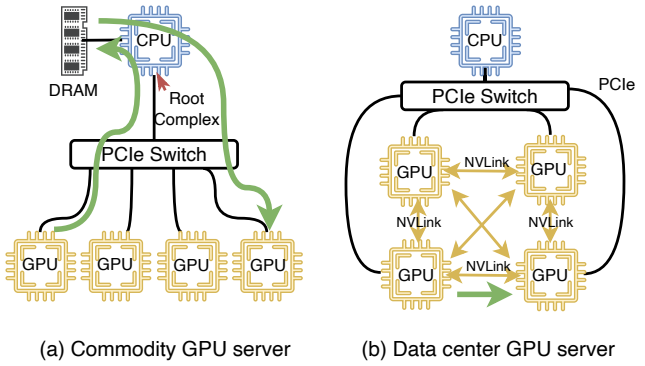

Figure 1: The difference in architecture between commodity and data center GPU servers. In data center servers, all GPUs are additionally connected via NVLink. The green arrows indicate the data transfer between two GPUs.

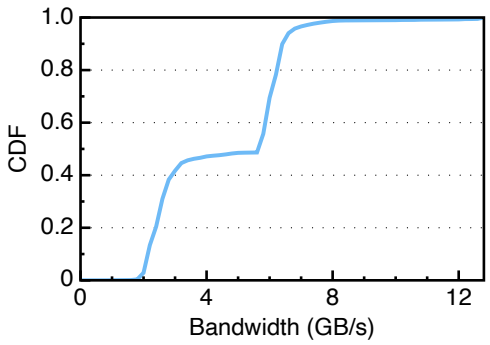

Figure 2: GPU communication bandwidth cumulative distribution function (CDF) of DeepSpeed when fine-tuning a 15B model. This experiment is performed on a server with 4×3090-Ti GPUs. Every two GPU share a CPU root complex.

to fine tune models. However, their communication resources are limited compared with data center training GPU server.

Opportunities: affordable and sufficient computational power. Compared with data center GPU servers, commodity GPU servers are more affordable for most people. For example, a DGX A100 with 8 fully connected A100 GPUs costs up to $200,000 [5]. Renting an EC2 P4 (with 8 × A100 GPUs) in the cloud needs $20,000 per month [2]. However, buying a commodity GPU server with 8 × 3090-Ti GPUs only needs $20,000. Although commodity GPU servers are cheaper, they provide sufficient computational power. For example, a 3090-Ti GPU offers twice the FP32 computing performance and similar tensor cores of an A100 GPU (Table 1).

Gaps: scarce communication resources. However, compared to data center GPU servers, commodity GPU servers' communication resources are limited. First, the bandwidth of inter-GPU communication in commodity GPU servers is low. In data center GPU servers, GPUs are fully connected via NVLink and NVLink Switch, which enables up to a 900 GB/s bandwidth between any two GPUs [12]. However, commodity GPU servers do not support fully-connective NVLink or NVLink Switch. Thus, their inter-GPU communication only reaches a bandwidth of PCIe bus (16 GB/s).

Second, communication contention is a hurdle to making every single GPU fully utilize communication resources. GPUDirect Peer to Peer (GPUDirect P2P), which enables GPU-to-GPU data operations directly over the memory fabric (PCIe, NVLink), is unavailable on commodity GPUs. Therefore, inter-GPU communication is first routed through CPU to DRAM, and then transferred to the target GPU. However, in most commodity GPU servers, multiple GPUs are connected to a CPU via a single PCIe Switch (Figure 1a). In this kind of GPU topology, if multiple GPUs transfer data at the same time, the communication is bounded by the shared the CPU root complex (pointed by the red arrow), and each GPU only utilizes a portion of the CPU root complex's bandwidth.

2.3 Analysis of DeepSpeed

The limited GPU memory capacity restricts the trainable model size both on commodity and data center GPU servers. The state-of-the-art fine-tuning system, Microsoft DeepSpeed, supports heterogeneous memory to train large-scale models on data center GPU servers (e.g., DGX-2 [4]). However, it is unsuitable for commodity GPU servers due to the communication problem mentioned in §2.2. Experimentally, we profile DeepSpeed with a GPT-like model on a 4×3090-Ti server and find that communication time accounts for over 70% of total training time; the detailed configuration of DeepSpeed is in the §4. We conclude that this is because the communication pattern of DeepSpeed is mismatched with the scarce communication resources on commodity GPU servers. In detail, there are two reasons as follows.

Massive communications of DeepSpeed, but low communication bandwidth on commodity GPU servers. DeepSpeed generates massive communications. Through profiling, we find that the communication traffic of DeepSpeed is horribly 7.3× of the model size in a single training step. Two factors contribute to this phenomenon. First, DeepSpeed is based on data parallelism, and thus needs to all-reduce gradients of every parameter across all GPUs to ensure parameter consistency. Second, DeepSpeed shards model parameters among GPUs. During training, it all-gathers parameters of each layer. Such frequent all-to-all collective communications incur minor overhead on data center GPU servers with GPUDirect P2P and NVLink enabled. However, with a commodity GPU server, the communication overhead will be fully exposed due to a low inter-GPU bandwidth.

Frequent all-to-all collective communications of DeepSpeed, but communication resources contention on commodity GPU servers. Figure 2 shows GPU communication bandwidth cumulative distribution function in one training step when fine-tuning a 15B model. We observe that most data communication of DeepSpeed only reaches 50% of the maximum bandwidth of the CPU root complex. The reason is that all GPU communications must go through the CPU's root complex due to the lack of GPUDirect P2P

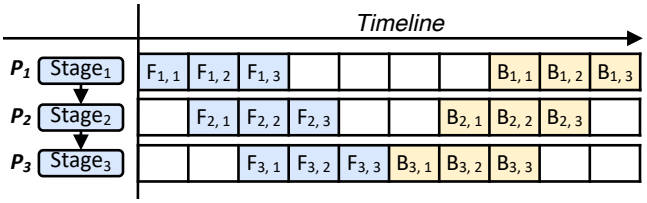

Figure 3: GPipe. Each stage contains multiple layers of a model. P_i denotes the i_{th} GPU. $F_{i,j}$ and $B_{i,j}$ denote the i_{th} stage's forward/backward execution on the j_{th} microbatch respectively. Each square denotes a time unit. The left side shows the order of activation transfer between stages during forward. The right side shows the GPipe execution in one step. The blank squares indicate computation bubbles.

on commodity GPU servers (as mentioned in §2.2). In DeepSpeed, there are massive all-to-all GPU collective communications, which make it frequent that multiple GPUs under the same CPU transfer data simultaneously. It leads to massive communications to contend the bandwidth of CPU's root complex.

In summary, although DeepSpeed enables large-scale model training on limited GPU memory by leveraging heterogeneous memory, scarcity of communication resources still hinders efficient large model fine-tuning on commodity GPU servers.

3 MOBIUS DESIGN

To enable communication-efficient large-scale model training on commodity GPU servers, we propose Mobius. Mobius is a novel *pipeline parallelism* to reduce communications, while enabling *heterogeneous memory* to train large-scale models (§3.1). To take full advantage of the Mobius pipeline's opportunities, Mobius proposes a model partition algorithm based on *mixed-integer programs (MIP)* to find the optimal partition scheme (§3.2). After partitioning, Mobius uses *cross mapping* to map stages to different GPUs and reduces communication contention (§3.3).

3.1 Mobius Pipeline

Traditional pipeline parallelism (e.g., GPipe in Figure 3) first partitions a model into some stages, each of which includes multiple layers of the model. It maps only one stage to each GPU, and divides a batch into multiple microbatches executed in the pipeline. However, traditional pipeline parallelism only utilizes GPU memory, making trainable model scale bounded by GPU memory capacity.

Different from traditional pipeline parallelism, Mobius pipeline leverages heterogeneous memory to train larger models without enough GPU memory. In Mobius pipeline, the number of stages can be more than that of GPU, and each GPU is responsible for multiple stages' execution. These stages are stored in DRAM. Mobius transfers a stage's copy from DRAM to GPU memory before executing it, and frees this copy in GPU after finishing the stages' execution on all microbatches. Note that we focus on extending GPU memory with only DRAM, since publicly available pretrained models can usually fit in DRAM and the limited bandwidth of SSDs is a performance bottleneck on a single server.

Figure 4a shows an example of Mobius pipeline. We assume that the model is divided into S *stages*, and these stages are mapped to N GPUs (e.g., $S = 8, N = 4$ and P_1, P_2, P_3, P_4 are GPUs in Figure 4). Each stage executes M microbatches in a training step, and M is equal to N (e.g., $M = 4$ in Figure 4). $Stage_{1,5}$ are mapped to P_1, $Stage_{2,6}$ are mapped to P_2, $Stage_{3,7}$ are mapped to P_3, and $Stage_{4,8}$ are mapped to P_4. During the forward, microbatch m_1 is first computed on $Stage_{1\sim4,}$, and when it reaches $Stage_4$, the forward of the last microbatch m_4 on $Stage_1$ is finished. At the time, $Stage_5$ can be transferred to GPU memory from DRAM and replaces $Stage_1$ on P_1. The activation of m_1 on $Stage_4$ is transferred to P_1, and P_1 continues to execute $Stage_5$'s forward function on m_1. Other microbatches' computing and stages' replacements are similar.

Low communication traffic. Here we analyze the communication traffic of Mobius and DeepSpeed *theoretically* to show that Mobius pipeline reduces communications traffic. We take the case of mixed precision training [30] for example.

In one training step of Mobius, only two copies of parameters need to be transferred to GPU memory in FP16 for forward and backward execution (P_{Mobius}). Besides, Mobius pipeline needs to offload activations from GPU memory to DRAM after forward and upload them from DRAM to GPU memory before backward (A_{Mobius}). At the end of each step, the parameters' gradients should be transferred to DRAM for parameter update (G_{Mobius}). Therefore, the communication traffic of Mobius is

$$\begin{aligned} P_{Mobius} &= 2 \times \frac{total\_parameter\_size}{2} \\ A_{Mobius} &= 2 \times total\_activation\_size \\ G_{Mobius} &= total\_gradient\_size \end{aligned}$$

$$\begin{aligned} &\text{CommunicationTraffic}_{Mobius} \\ &\quad = P_{Mobius} + C_{Mobius} + G_{Mobius} \\ &\quad \approx 1.5 \times total\_parameter\_size \end{aligned} \tag{1}$$

In comparison, DeepSpeed transfers two copy of the parameters in FP16 from DRAM to GPUs, and transfers $2 \times (N-1)$ copies of the parameters between all GPUs ($P_{DeepSpeed}$). At the same time, the activations need to be transferred between DRAM and GPU memory twice as much as Mobius ($A_{DeepSpeed}$). In backward, each GPU generates a version of gradients ($G_{DeepSpeed}$). Therefore, gradients in each GPU need to be first all-reduced and then swapped to DRAM for parameter update. In summary, the communication traffic of DeepSpeed is

$$\begin{aligned} P_{DeepSpeed} &= 2N \times \frac{total\_parameter\_size}{2} \\ A_{DeepSpeed} &= 2 \times total\_activation\_size \\ G_{DeepSpeed} &= N \times total\_gradient\_size \end{aligned}$$

$$\begin{aligned} &\text{CommunicationTraffic}_{DeepSpeed} \\ &\quad = P_{DeepSpeed} + C_{DeepSpeed} + G_{DeepSpeed} \\ &\quad \approx 1.5N \times total\_parameter\_size \end{aligned} \tag{2}$$

If we use checkpoint and recomputation [17] in fine-tuning, $total\_activation\_size$ is negligible. Each parameter's gradient size is equal to half of parameter's size, due to using FP16 training mode. According to Equation 1 and 2, Mobius can reduce communication traffic by $N\times$ (N is the number of GPUs).

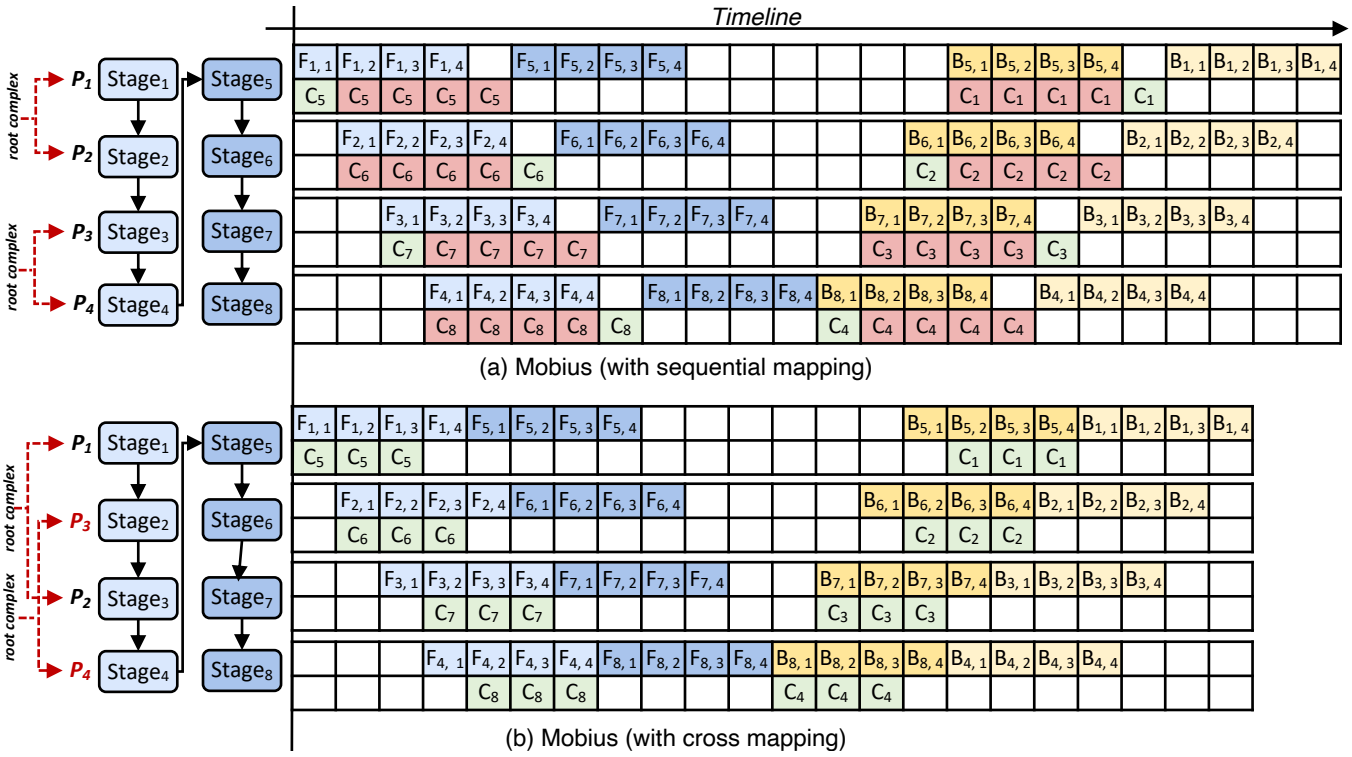

Figure 4: Mobius pipeline. P_i denotes the i_{th} GPU. The two GPUs pointed by the red dashed arrows share the bandwidth of the same CPU root complex. $F_{i,j}$ and $B_{i,j}$ denote the i_{th} stage's forward/backward execution on the j_{th} microbatch respectively. C_i denotes the data transfer of the i_{th} stage from DRAM to GPU memory. The light green squares indicate no communication contention, while the dark red squares indicate communication contention during data transfer.

Convergence discussion. Mobius pipeline updates parameters in the same way as GPipe, which uses synchronous parameter's gradient update instead of asynchronous parameter update in PipeDream [31], so it can ensure the same model convergence and accuracy performance as without pipelining.

3.2 Model Partition

Model partition has always been an important problem, which determines the overall training throughput in pipeline parallelism. Although there are existing pipeline model partition formulations or algorithms [24, 31], they do not work for Mobius since they only consider the case where models are only stored in GPU memory.

Mobius's pipeline parallelism with heterogeneous memory is complex. To model it, our partition algorithm must consider these additional factors not covered in traditional pipeline parallelism:

- Multi stages. In Mobius pipeline, each GPU processes multiple stages, and stages are swapped between the GPU and DRAM. Thus, both computation and communication time contribute to the stage execution time in Mobius pipeline. More complicated, some communication time can be hidden by computation.
- Prefetching. Mobius reserves a portion of GPU memory for prefetching the next stage's data to overlap communication with computation. In a dilemma, too much reserved GPU memory causes small stages, incurring more activations communication among GPUs, while too little reserved GPU memory limits the prefetch of the next stage, under-utilizing computation to overlap communication. Therefore, the algorithm should consider memory allocation of prefetching carefully.

Model partition algorithm. The model partition problem is equivalent to how to assign each model layer to different stages. To formulate this problem, we use boolean variables $B_{i,j}$ to denote whether the i_{th} model layer is placed in the j_{th} stage, where $1 \le i \le L$, $1 \le j \le L$ (L is the number of model layers). Note that we do not know the stage count beforehand, but the maximum stage count is L. Thus, we allocate L logical stages for convenience; for a given j, if all $B_{:,j}$ equals 0, it means the j_{th} stage does not exist physically. Our goal is to find a group of $B_{i,j}$ to minimize the training time of one step, considering memory limitation and pipeline order.

We employ a mixed-integer program (MIP) to find the best group of $B_{i,j}$. Table 2 summarizes all used variables. The objective is to minimize the training time of a step (i.e., the start time of executing the first stage's backward function on the last microbatch, $t^b_{0,M}$, plus with its backward duration T^b_0), which can be formulated as:

$$\begin{aligned} &minimize && t^b_{0,M} + T^b_0 \\ &subject\ to && Memory\ constraints \\ & && Pipeline\ order\ constraints \end{aligned} \tag{3}$$

Table 2: Variables used in MIP partition algorithm. Optimization variables $B_{i,j}$ are the searching space. Intermediate variables can be computed if we know values of $B_{i,j}$. In intermediate variables, $e \in \{f, b\}$, f means forward function, and b means backward function.

Constant variables:	
L	Number of the model's layers
N	Number of GPUs
M	Number of microbatches
G	Per-GPU memory capacity
B	Average GPU communication bandwidth
Optimization variables:	
$B_{i,j}$	Boolean variables. If $B_{i,j}$ is true, it means i_{th} model layer is in j_{th} stage.
Intermediate variables:	
m_i	i_{th} microbatch
s_i	i_{th} stage
a^i	Activation size of s_i
g^i	Activation gradient size of s_i
$t^e_{i,j}$	Start time of s_i's function e on m_j
T^e_i	Duration of s_i's function e on a microbatch
D^e_i	Duration of s_i finishes e on M microbatches
S^e_i	GPU memory required by s_i's function e
R^e_i	Reserved GPU memory in s_i's function e
P^e_i	Prefetch data size of s_i in function e

During model's training, two types of constraints need to be satisfied, namely memory constraints and pipeline order constraints.

Memory constraints: the data stored in the GPU should not exceed the GPU's memory. First, the GPU memory should hold current computing stage's parameters and the intermediate data during training. This constraint is formulated as follows:

$$S^e_j \le G, \quad j \in [1, L], e \in \{f, b\} \tag{4}$$

Second, except for the first stage in the forward and the last stage in the backward, the data of the next stage need to be prefetched. The amount of data prefetched for the next stage cannot exceed the reserved GPU memory. The constraints are formulated as follows:

$$\begin{cases} P^f_j \le G - S^f_{j-N}, & j \in (N, L] \\ P^b_j \le G - S^b_{j+N}, & j \in [1, L-N] \end{cases} \tag{5}$$

Third, prefetch should finish before the current computing stage finishes forward or backward on all microbatches, and the size of the prefetched data should not exceed the size of the next stage (Constraint 6).

$$\begin{cases} P^f_j \le B \times D^f_{j-N}, & j \in (N, L] \\ P^b_j \le B \times D^b_{j+N}, & j \in [1, L-N] \\ P^e_j \le S^e_j, & j \in [1, L], e \in \{f, b\} \end{cases} \tag{6}$$

D^e_i is the total time that s_i finishes e function on all M microbatches. It can be presented by the start time of the first and last microbatch execution (Equation 7).

$$D^e_j = T^e_j + t^e_{j,M} - t^e_{j,1}, \quad j \in [1, L], e \in \{f, b\} \tag{7}$$

Pipeline order constraints: the execution of each stage in the pipeline is dependent. First, Constraint 8 formulates that each stage needs to receive the computation results of the adjacent stages before it starts execution. In forward, each stage (except the first one) needs to receive the activation of the previous stage. In backward, each stage (except the last one) needs to receive the activation gradient of the latter stage. After a stage finishes forward or backward on a microbatch, the activation or the activation gradient of this microbatch should be transferred to the GPU which stores the next stage.

$$\begin{cases} t^f_{j,m} \ge (t^f_{j-1,m} + T^f_{j-1}) + \dfrac{a_{j-1}}{B}, & j \in (1, L], m \in [1, M] \\ t^b_{j,m} \ge (t^b_{j+1,m} + T^b_{j+1}) + \dfrac{g_{j+1}}{B}, & j \in [1, L), m \in [1, M] \end{cases} \tag{8}$$

Second, Constraint 9 formulates that a stage can start computation only after the data of this stage is in GPU memory. If Mobius fails to prefetch all data of this stage, the computation will be blocked until data is all uploaded to GPU memory.

$$\begin{cases} t^f_{j,1} \ge (t^f_{j-N,M} + T^f_{j-N}) + \dfrac{S^f_j - P^f_j}{B}, & j \in (N, L] \\ t^b_{j,1} \ge (t^b_{j+N,M} + T^b_{j+N}) + \dfrac{S^b_j - P^b_j}{B}, & j[1, L-N] \end{cases} \tag{9}$$

Third, Mobius executes the microbatches on the same stage sequentially. Each GPU can only execute one stage's forward or backward function on a microbatch at a time (Constraint 10).

$$\begin{aligned} t^e_{j,m} &\ge t^e_{j,m-1} + T^e_j, \\ where \quad & j \in [1, L], m \in (1, M], e \in \{f, b\} \end{aligned} \tag{10}$$

Forth, the backward of a step begins after the forward finishes (Constraint 11).

$$t^b_{L,1} \ge t^f_{L,M} + T^f_L \tag{11}$$

Profiling. MIP partition algorithm requires the pre-knowledge of the memory footprint and computing time of each layer. A basic way to get this information is to profile the whole model and collect each layer's statistics, which is slow since prefetching is disabled for more accurate statistics. Mobius leverages the model layer similarity to reduce the profiling time. There are a large number of identical layers in large-scale models (e.g., Transformer blocks in GPT-3). These layers share similar GPU memory footprint and computing time. Mobius merges a group of equal layers into one based on the model layer similarity. This compresses a model to a smaller one, enabling profiling to be completed in less time.

Solving MIP. We solve this MIP by using Gurobi Optimizer [10] to obtain a balanced partition. The solving time only costs up to several seconds in our evaluation, which is negligible compared to the overall fine-tuning duration (hours to days).

3.3 Cross Mapping

After the model partition, Mobius needs to map each stage to a GPU. The stage mapping needs to consider communication contention. We observe that when mapping adjacent stages to the

GPUs under the same CPU root complex, the Mobius pipeline's performance suffers from communication contention. For example, in Figure 4a, $Stage_5$ and $Stage_6$ are mapped to P_1 and P_2, which share the bandwidth of the same CPU root complex. There is terrible communication contention when prefetching them (red squares of C_5 and C_6 in Figure 4a). As a consequence, it increases the time of data transfer and introduces more computation bubbles, which slows down the overall throughput of Mobius pipeline.

Based on the observation, Mobius maps adjacent stages to GPUs not under the same CPU root complex as much as possible, called *cross mapping*. Cross mapping brings larger time difference to upload stage data, which significantly reduces communication contention.

However, there are a lot of different cross mapping schemes. To find the best one, Mobius uses Equation 12 to estimate the contention degree between two stages, where $shared(i, j)$ is the number of GPUs under the same CPU root complex where $Stage_i$ and $Stage_j$ are located. If the GPUs storing $Stage_i$ and $Stage_j$ are under different CPU root complex, $shared(i, j)$ is zero. The more GPUs in the same root complex (a larger $shared(i, j)$), or the smaller the difference between j and i (a smaller $|i - j|$), the more likely it is to conflict.

$$contention(Stage_i, Stage_j) = \frac{shared(i, j)}{|i - j|} \tag{12}$$

Mobius searches all mapping schemes and finds the one with the minimal contention degree (Equation 13) as the best solution.

$$\min \sum_{i<j} contention(Stage_i, Stage_j) \tag{13}$$

Although cross mapping tries to avoid communication contention, there still may be multiple prefetches under the same root complex simultaneously. Mobius assigns higher priority to the prefetch of the stage that starts earlier, which further reduces communication contention. In implementation, Mobius uses cudaStreamCreateWithPriority API to assign priorities.

Figure 4b shows a concrete illustration of cross mapping. P_1 and P_2 share the bandwidth of the same CPU root complex, P_3 and P_4 share another one. We take $Stage_5$ and $Stage_6$ as an example. Instead of mapping $Stage_5$ and $Stage_6$ to P_1 and P_2, they are cross mapped to P_1 and P_3. Therefore, the data transfer of $Stage_5$ and $Stage_6$ can fully utilize the maximum bandwidth of the CPU complex without any communication contention. As an illustration of the prefetch priority, If the prefetching operations of $Stage_5$ and $Stage_7$ execute simultaneously, $Stage_5$ has a higher priority, because $Stage_5$ executes earlier. By using cross mapping, Mobius fully utilizes the bandwidth of different CPU root complexes. Compared with the Mobius with sequential mapping, it reduces 2 time units one training step in this example.

4 EVALUATION

In this section, we first demonstrate the end-to-end performance of Mobius, and then show the effectiveness of each design. We finally benchmark Mobius's scalability and its performance on a data center GPU server.

Setup. We use two setups. The first setup contains a server equipped with 1.5TB DRAM, two Intel Xeon Gold 6130 CPUs and 8×3090-Ti GPUs (each GPU has 24 GB memory). Every 4 GPUs are connected to a CPU root complex via PCIe 3.0x8 and a PCIe Switch. The second setup, referred to as data center GPU server, is an Amazon EC2 P3.8xlarge instance [1], which provides 4×V100 GPUs (16 GB memory) and enables GPUDirect P2P via NVLink with bandwidth of 300 GB/s. Unless specified, experiments use the first setup.

Table 3: Model configuration.

Number of parameters (billion)	Attention heads	Hidden dimension size	Number of layers	Microbatch size
3	32	2048	64	2
8	32	4096	40	2
15	64	5120	40	1
51	80	9216	50	1

GPU topologies. We evaluate Mobius on three GPU topologies, namely Topo 4, Topo 2+2, and Topo 1+3, to simulate different GPU allocations in a shared server. Topo 4 denotes four GPUs share a CPU root complex, Topo 2+2 denotes every two GPUs share one, and Topo 1+3 denotes three of four GPUs share one. Among them, Topo 2+2 has the least communication contention, while Topo 4's communication contention is the most severe.

Baselines. Our baselines are GPipe [26] and DeepSpeed [3]. GPipe is a kind of pipeline parallelism which trains models only using GPU memory. DeepSpeed is, to our knowledge, the state-of-art to train large-scale models using heterogeneous memory. DeepSpeed also supports pipeline parallelism only using GPU memory. We configure DeepSpeed in these two configurations. In the first one, it is configured in ZeRO-3 mode with heterogeneous memory (GPU memory and DRAM) enabled. In the second one, it is configured in pipeline parallelism. DeepSpeed with pipeline parallelism runs out of memory when training large-scale models, which do not fit in GPU memory. Therefore, DeepSpeed uses the first configuration unless specified.

Workloads. For the performance evaluation, we use GPT-like Transformer based models with different hidden dimensions and number of layers in Table 3. The sequence length is fixed to 512. The 3B model with batch size of 2 is the largest model that GPipe and DeepSpeed with pipeline parallelism can train. The Transformer block with a 9216 hidden dimension is the largest block a single GPU can hold during training. For convergence analysis, we use the GPT-2 model from [6], and fine tune this model on the WikiText2 [29]. We use the mixed precision training in all evaluations.

4.1 Overall Evaluation

We compare the per-step training time of GPipe, DeepSpeed and Mobius in Figure 5. In this experiment, we train all four models in Table 3 with a batch size of one on three GPU topologies. We have the following observations. 1) Both Mobius and DeepSpeed with heterogeneous memory are able to train larger models with larger batches compared with GPipe and DeepSpeed with pipeline parallelism. When increasing the model size, GPipe and DeepSpeed with pipeline parallelism run out of memory (OOM). 2) Mobius decreases

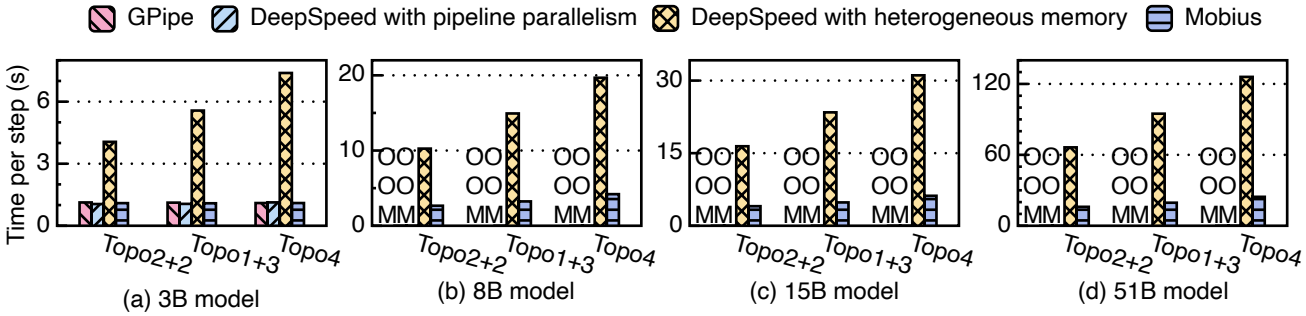

Figure 5: Per-step time of GPipe, DeepSpeed and Mobius, with different models and GPU topologies.

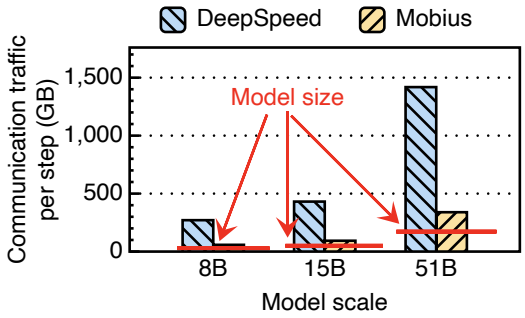

Figure 6: Communication traffic of DeepSpeed and Mobius. The red line denotes the size of model parameters.

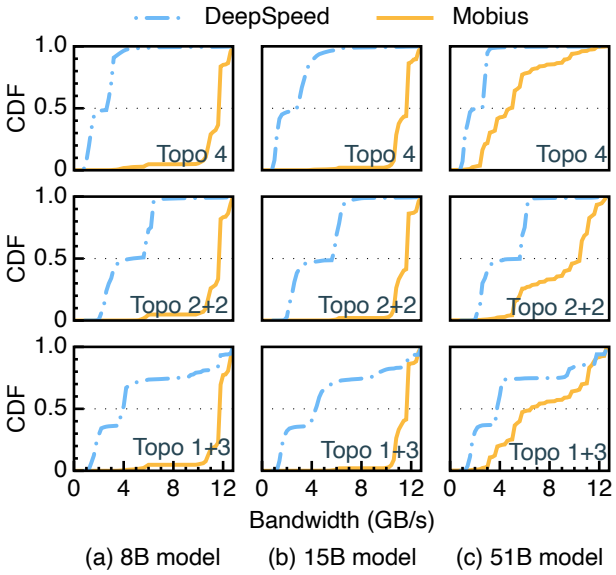

Figure 7: GPU communication bandwidth cumulative distribution function. DeepSpeed and Mobius trains these models using 4 GPUs with three different GPU topologies.

per-step training time by 3.8-5.1× compared with DeepSpeed with heterogeneous memory. This validates the effectiveness of Mobius's designs. 3) Mobius brings more significant performance improvement compared to DeepSpeed with heterogeneous memory when the GPU topology has more severe communication contention. This is because DeepSpeed with heterogeneous memory suffers more from communication contention on commodity GPU servers, while Mobius alleviates it with a careful partition and mapping algorithm. 4) Mobius keeps almost stable performance under different GPU topologies, thanks to the cross mapping mechanism.

4.2 Communication Analysis

To verify that Mobius solves the communication problems on commodity GPU servers, we collect communication traffic, bandwidth statistics, and non-overlapped communication cost during training.

Communication traffic. Figure 6 illustrates that DeepSpeed needs to transfer 7.3× the data of a model, while Mobius only transfers about 1.8× the data of a model. The reason is that DeepSpeed requires frequent GPU all-to-all collective communications to all-gather parameters and all-reduce gradients, while Mobius pipeline only transfers small activations and activation gradients. The result is consistent with the analysis in §3.1.

Bandwidth statistics. Figure 7 shows the cumulative distribution function of GPU communication bandwidth statistics in one training step. In Mobius more than half of the data is transferred at a bandwidth of more than 12 GB/s (the maximum bandwidth measured is 13.1 GB/s). However, DeepSpeed transfers most data at a bandwidth of less than 6 GB/s, which is half of the maximum bandwidth of the CPU root complex due to serious communication contention. Thus, Mobius effectively mitigates communication congestion on commodity GPU servers.

Non-overlapped communication time. Figure 8 exhibits the proportion of communication time non-overlapped by calculation in per-step training time. We have the following observations. 1) Compared with DeepSpeed, Mobius reduces the proportion of non-overlapped communication time by up to 46%. This verifies that the designs of Mobius can make full use of computation to overlap communication overhead. Smaller proportion of non-overlapped communication in Mobius time also implies less computation stall time due to communication. 2) Mobius overlaps communication overhead better under Topo 2+2. This is because cross mapping technology helps Mobius make better use of the topology information to reduce communication overhead.

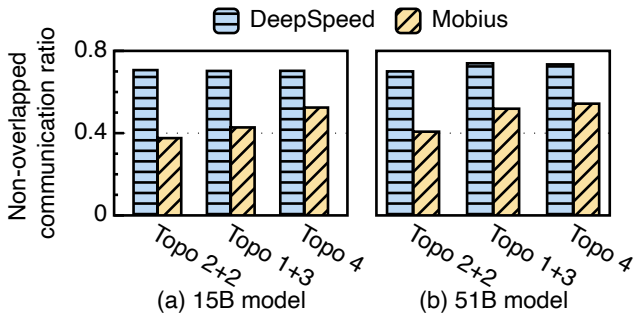

Figure 8: The proportion of non-overlapped communication in DeepSpeed and Mobius.

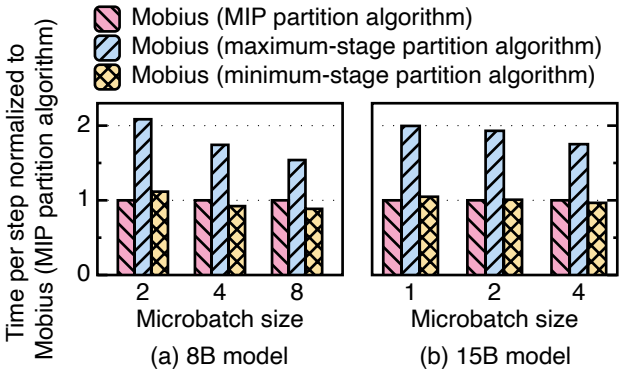

Figure 9: Per-step training time with different model partition algorithms. Mobius trains three kinds of models with different hidden dimension sizes and using different batch sizes. These experiments are done using 4 GPUs, and every two GPUs share a CPU root complex on a single GPU server.

From Figure 6, 7 and 8, we can conclude that Mobius's performance improvement over DeepSpeed comes from the reducing of communication traffic, alleviating communication contention and overlapping communication overhead by computation.

4.3 Effect of MIP Partition Algorithm

To evaluate the effectiveness of MIP partition algorithm, we compare the following three different model partition mechanisms. In this experiment, Mobius trains three kinds of models with different hidden dimension sizes and using different microbatch sizes. We train these models using the GPU topology Topo 2+2.

- MIP partition algorithm. This is our proposed algorithm (described in §3.2).
- Maximum-stage partition algorithm. Each stage contains as many Transformer blocks as possible without running out of memory.
- Minimum-stage partition algorithm. Each stage contains only one Transformer block.

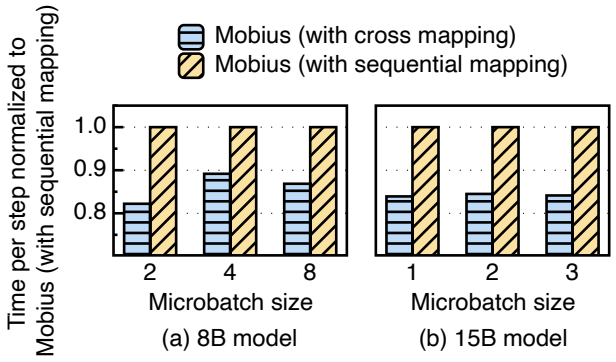

Figure 10: Per-step training time comparison between different stage mapping mechanisms of Mobius. The time is normalized to Mobius (with sequential mapping).

Figure 9 shows the training time with different partition algorithms (normalized to MIP partition algorithm). MIP partition algorithm can reduce training time by up to 51% compared to other algorithms, which illustrates that a more balanced partition scheme generated by MIP partition algorithm could significantly improve the training performance. Specifically, we have the following three observations. 1) In most cases, maximum-stage partition algorithm has the worst performance. It fills GPU memory with the current computing stage's data, which prevents prefetching the next stage's data and eliminates the opportunity of overlapping communication by computation. 2) When Transformer blocks and microbatches are large, MIP partition algorithm and minimum-stage partition algorithm draw a similar performance. This is because that, in this case, a single GPU's memory can store only one layer's parameters and computing data, which makes the partition scheme generated by MIP partition algorithm is exactly the same as that of minimize-stage partition algorithm. 3) When Transformer blocks and microbatches are small, computation overhead is slight, and the overhead of frequent activation and activation gradient transfer between GPUs becomes significant. In this case, the solution generated by MIP partition algorithm is more efficient.

4.4 Effect of Cross Mapping

We evaluate cross mapping's performance improvement by using sequential mapping as the baseline and keeping all other components of Mobius the same. Sequential mapping mechanism maps stages to GPUs according to the number of GPUs without considering the PCIe topology of GPUs. In this experiment, Mobius trains two models with different batch sizes. We train these models using 8 GPUs with the topology where every four GPUs share a CPU root complex.

Figure 10 shows the training time per step of two mapping mechanisms (normalized to sequential mapping). 1) Cross mapping reduces per-step training time by 11.3%-18.1% compared with sequential mapping, validating its design. 2) The performance improvement brought by cross mapping is less significant when the size of Transformer blocks and microbatch becomes large. This is because that extremely large microbatches and Transformer blocks

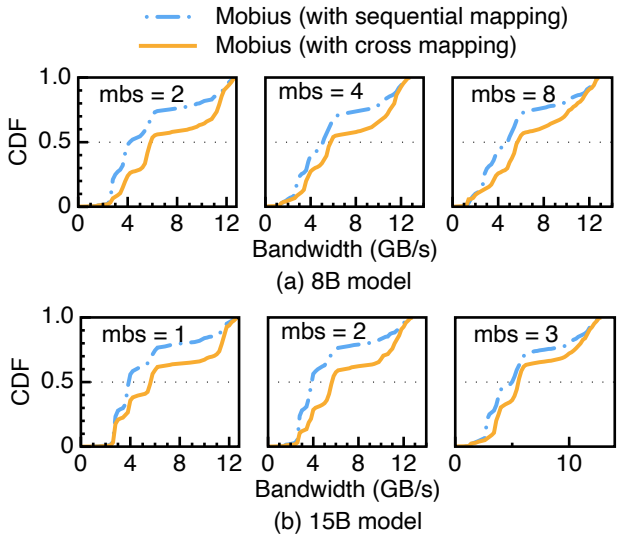

Figure 11: GPU communication bandwidth cumulative distribution function per step during training using different stage mapping mechanisms. Mobius trains these models with different microbatch sizes (mbs).

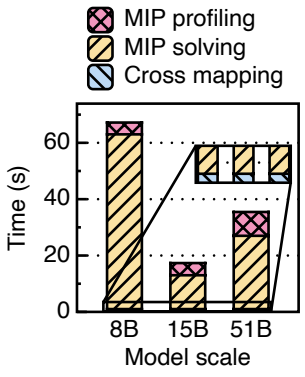

Figure 12: MIP algorithm and cross mapping overhead.

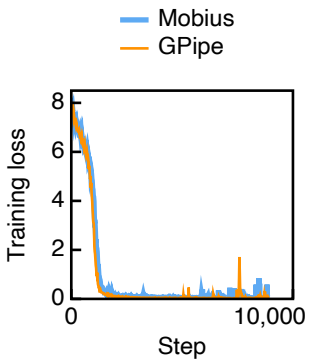

Figure 13: Training loss of Mobius and GPipe.

result in more computing time, which outweighs the communication time and makes the reduced communication time of cross mapping insignificant.

To verify the effectiveness of cross mapping on communication contention, similar to the setting of Figure 7, we collect GPU communication bandwidth statistics in one training step; see Figure 11. Compared to using sequential mapping, more data is transferred in a higher bandwidth when using cross mapping, which shows cross mapping's superiority for mitigating communication contention.

4.5 Mobius Overhead

To analyze the extra overhead introduced by Mobius, we profile model partition and cross mapping overhead during training in Topo 1+3. The model partition overhead is contributed by profiling and MIP solving. From Figure 12, we have the following observations. 1) These extra overheads are negligible compared to the overall fine-tuning overhead (hours to days). 2) Although the 15B model is larger than the 8B model, they have close profiling time. This is because only the different layers need to be profiled after leveraging model layer similarity, which makes profiling time only relate to the computation time of different layers, These two models have similar hidden dimension sizes, leading to similar computation time of different layers, thus having close profiling time. 3) When the hidden dimension size of a model is small, the MIP solving brings high overhead. The reason is the maximum number of layers that can be stored in GPU is large, increasing the search space. More layers in models also bring higher overhead, because there are more variables corresponding to these layers in the MIP.

4.6 Convergence Evaluation

Figure 13 shows the comparison of training loss curves between GPipe and Mobius when fine tuing GPT-2 model on WikiText-2. We use 8 × 3090-Ti GPU when using GPipe and 4 × 3090-Ti GPU when using Mobius. We observe the training loss curves of the GPipe and Mobius are almost overlapped. The result validates that Mobius does not hurt convergence like GPipe[24] ,which is consistent with the analysis in §3.1. The slight difference between the curves of Mobius and GPipe is due to the variation of randomness caused by different numbers of GPUs.

4.7 Scalability Evaluation

To analyze the scalability of Mobius, we train the 15B model by sweeping the number of GPUs from 2 to 8. For all configurations, each half of the GPUs shares a separate CPU root complex. We constantly set the microbatch size to 1 and increase the batch size with increasing number of GPUs. Figure 14 shows that Mobius exceeds perfect linear scaling. When the GPUs cannot be divided equally into two groups, Mobius has a slight performance degradation, due to the uneven communication contention under the two CPU root complexes.

4.8 Evaluation on Data Center GPU Server

Although Mobius is designed for commodity GPU servers, we also evaluate its performance on the data center GPU server to test Mobius's sensitivity to different server configurations. In this experiment, we train the 8B model and the 15B model with microbatch size of 2 using DeepSpeed and Mobius. Note that data center GPU servers are equipped with NVLink and GPUDirect P2P, and thus the inter-GPU communication bandwidth is sufficient.

Figure 15a illustrates the performance and per-step price of Mobius and DeepSpeed on the data center GPU server and the 3090-Ti GPU server. We observe that: 1) DeepSpeed and Mobius both have performance improvement on the data center GPU server. This is because the NVLink on the data center GPU server reduces the communication overhead between GPUs. 2) DeepSpeed needs frequent collective communication between GPUs as we analyze in

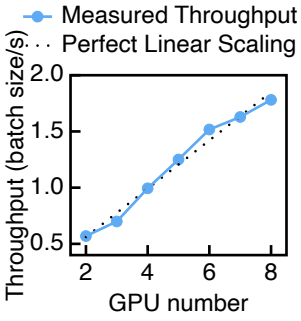

Figure 14: Linear scalability of Mobius on the commodity GPU server.

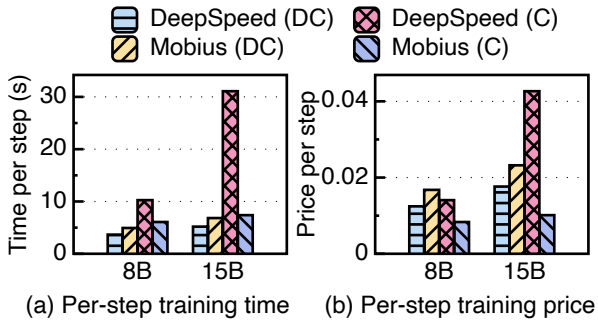

Figure 15: Performance and per-step price of Mobius and DeepSpeed. DC denotes data center GPU server, and C denotes commodity GPU server.

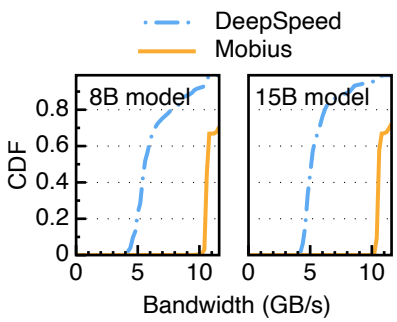

Figure 16: GPU-CPU communication bandwidth CDF on the data center GPU server.

§2.3, but there is only a small number of activations and activation gradients transferred between GPUs in Mobius. Therefore, DeepSpeed has more significant performance improvement on the data center GPU server. 3) Mobius performs worse than DeepSpeed on the data center GPU server. This is because the aggregate communication bandwidth of Mobius is less. Mobius only utilizes the NVLinks of GPUs storing adjacent stages, while DeepSpeed fully utilizes fully-connected NVLinks for all-to-all communications.

We collect GPU-to-CPU and CPU-to-GPU communication bandwidth statistics in one training step, shown in Figure 16. Compared with the commodity GPU server case, the communication contention gap between DeepSpeed and Mobius is reduced, due to the reduction of collective communication overhead in DeepSpeed. However, the communication contention in Mobius is still lower. This is because there is fewer stage data transfer at the same time in Mobius pipeline.

We calculate the per-step training price according to [1] and [8]. From Figure 15b, compared with using DeepSpeed on the data center GPU server, the per-step training time increases by 42% when using Mobius on 3090-Ti GPU server, but the price per step decreases by 43%. Mobius trades small training performance decreasing for a much lower training price.

5 RELATED WORK

In recent years, there have been a wealth of works on large model training systems. We classify these works into two categories: scale-up, and scale-out methods.

Scale-out methods. To satisfy the memory requirement of large model training, several works use multiple GPUs to increase the aggregated memory of GPUs. Pipeline parallelism and model parallelism are the most widely used scale-out training methods. Pipeline parallelism [14, 19, 20, 24, 27, 31, 32, 45] partitions a model in the unit of layers (vertical partition). Model parallelism [39, 40, 43] splits each layer into multiple GPUs (horizontal partition). Pipeline parallelism has less communication overhead than model parallelism. However, pipeline parallelism is difficult to guarantee high GPU utilization and model convergence at the same time [33]. ZeRO [35] is a recent work for scale-out training. ZeRO splits a model into multiple GPUs' memory and uses collective communications to gather each model layer [28, 44], trading communication for the reduction of GPU memory consumption. Collectively, the trainable model scale of these systems is bounded by the aggregated GPU memory capacity.

Scale-up methods. Scale-up methods break the GPU memory limit to train larger models by leveraging external storage resources such as DRAM and SSD. These works [15, 23, 34, 42] swap model data between GPU and external storage based on computation graph. They focus on extending the memory capacity of a single GPU, but ignoring communication issues in the multi-GPU scenario, which is more common for training large-scale models. ZeRO-Offload [37] stores optimizer states and gradients in DRAM while maintaining model parameters in each GPU's memory. Due to the redundant copies of model parameters, the model scale is limited by a single GPU's memory capacity when using ZeRO-Offload. To enhance the trainable model scale, ZeRO-Infinity [36] distributed stores model states in multiple GPUs to reduce redundancy. Besides, ZeRO-Infinity extends the offloading storage to NVMe devices and offloads all model states and activations to the external storage. The works of the ZeRO family trade frequent communication collectives for less redundant copies of model parameters in GPU memory, and also require frequent communication collectives to keep model states' consistency. They assume the communication bandwidth is sufficient and GPUDirect P2P is enabled. However, it is a common situation that communication resources are scarce on commodities GPU servers. In this situation, the training performance of ZeRO family suffers. ZeRO family is integrated in DeepSpeed [3].

Different from existing works [14, 19] on commodity GPUs based on pipeline parallelism, Mobius enables heterogeneous memory, which increases trainable model scale. Besides, compared with prior scale-up methods, Mobius focuses on the communication problem when enabling heterogeneous memory in the multi-GPU scenario. Mobius brings less communication cost than ZeRO family, and is PCIe topology-aware to make full use of communication resources on GPU servers. It also leverages carefully scheduling to overlap communication overhead by computation. Therefore, Mobius is more communication-friendly, thus more suitable when communication resources are scarce.

6 CONCLUSION

We present Mobius, a system for efficient large-scale model fine tuning on commodity GPU servers. Mobius introduces a novel pipeline parallelism, which enables heterogeneous memory to train larger model using limited GPU memory while brings fewer communications. To take full advantage of Mobius pipeline, Mobius proposes a mixed-integer-programs based partition algorithm to find an optimal model partition solution, which balances computation and communication. Mobius employs the cross mapping technique to map stages to GPUs with minimum communication contention. We demonstrate the efficiency and effectiveness of Mobius's designs with experiments on a variety scale of deep learning models and GPU servers with different topologies. The results show that Mobius fares better than the state-of-the art.

ACKNOWLEDGMENTS

We sincerely thank our shepherd and the anonymous reviewers for their valuable feedback. This work is funded by the National Natural Science Foundation of China (Grant No. 61832011) and Open Research Program of Zhejiang Lab (No. 2020KC0AB03).

REFERENCES

[1] [n. d.]. Amazon EC2 P3 Instances. https://aws.amazon.com/ec2/instance-types/p3/.
[2] [n. d.]. Amazon EC2 P4 Instances. https://aws.amazon.com/ec2/instance-types/p4/.
[3] [n. d.]. DeepSpeed. https://github.com/microsoft/DeepSpeed.
[4] [n. d.]. DGX-2. https://www.nvidia.com/en-us/data-center/dgx-2/.
[5] [n. d.]. DGX A100. https://www.nvidia.com/en-us/data-center/dgx-a100/.
[6] [n. d.]. EleutherAI/gpt-j-6B. https://huggingface.co/EleutherAI/gpt-j-6B.
[7] [n. d.]. GEFORCE RTX 3090 Family. https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/.
[8] [n. d.]. GPU cloud servers. https://en.immers.cloud/gpu/.
[9] [n. d.]. GPUDirect. https://developer.nvidia.com/gpudirect.
[10] [n. d.]. Gurobi. https://www.gurobi.com.
[11] [n. d.]. NVIDIA A100 TENSOR CORE GPU. https://www.nvidia.com/en-us/data-center/a100/.
[12] [n. d.]. NVLink and NVSwitch. https://www.nvidia.com/en-us/data-center/nvlink/.
[13] [n. d.]. OpenAI's GPT-3 Language Model: A Technical Overview. https://lambdalabs.com/blog/demystifying-gpt-3/.
[14] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. 2022. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 472–487.
[15] Jonghyun Bae, Jongsung Lee, Yunho Jin, Sam Son, Shine Kim, Hakbeom Jang, Tae Jun Ham, and Jae W Lee. 2021. FlashNeuron:SSD-Enabled Large-Batch Training of Very Deep Neural Networks. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 387–401.
[16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[17] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).
[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
[19] Saar Eliad, Ido Hakimi, Alon De Jagger, Mark Silberstein, and Assaf Schuster. 2021. Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 381–396. https://www.usenix.org/conference/atc21/presentation/eliad
[20] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. 2021. 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*. 431–445.
[21] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. https://doi.org/10.48550/ARXIV.2101.03961
[22] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 328–339. https://doi.org/10.18653/v1/P18-1031
[23] Chien-Chin Huang, Gu Jin, and Jinyang Li. 2020. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1341–1355.
[24] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019).
[25] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).
[26] Chiheon Kim, Heungsub Lee, Myungryong Jeong, Woonhyuk Baek, Boogeon Yoon, Ildoo Kim, Sungbin Lim, and Sungwoong Kim. 2020. torchgpipe: On-the-fly Pipeline Parallelism for Training Giant Models. (2020). arXiv:2004.09910
[27] Shigang Li and Torsten Hoefler. 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.
[28] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).
[29] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).
[30] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740* (2017).
[31] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 1–15.
[32] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*. PMLR, 7937–7947.
[33] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. 1–15.
[34] Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, and Xuehai Qian. 2020. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 891–905.
[35] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 262–277.
[36] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.
[37] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 551–564.
[38] Or Sharir, Barak Peleg, and Yoav Shoham. 2020. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900* (2020).
[39] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems* 31 (2018).
[40] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).
[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[42] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. 2018. Superneurons: Dynamic GPU memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*. 41–53.

[43] Minjie Wang, Chien-chin Huang, and Jinyang Li. 2019. Supporting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–17.

[44] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE transactions on Big Data* 1, 2 (2015), 49–67.

[45] Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher Aberger, and Christopher De Sa. 2021. Pipemare: Asynchronous pipeline parallel dnn training. *Proceedings of Machine Learning and Systems* 3 (2021), 269–296.

[46] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

Received 2022-07-07; accepted 2022-09-22