MixPipe: Efficient Bidirectional Pipeline Parallelism for Training Large-Scale Models

Weigang Zhang[1,2], Biyu Zhou[1*], Xuehai Tang[1,2], Zhaoxing Wang[1,2], Songlin Hu[1,2]
[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]School of Cyber Security, University of Chinese Academy of Sciences
{zhangweigang, zhoubiyu, tangxuehai, wangzhaoxing, husonglin}@iie.ac.cn

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247730

***Abstract*—The rapid development of large-scale deep neural networks has put forward an urgent demand for the efficiency of parallel training. Recently, bidirectional pipeline parallelism has been recognized as an effective approach for improving training throughput. This paper proposes *MixPipe*, a novel bidirectional pipeline parallelism for efficiently training large-scale models in synchronous scenarios. Compared with previous proposals, *MixPipe* achieves a better balance between pipeline utilization and device utilization, which benefits from the flexible regulating for the number of micro-batches injected into the bidirectional pipelines at the beginning. *MixPipe* also features a mixed schedule to balance memory usage and further reduce the bubble ratio. Evaluation results show that: for Transformer based language models (i.e., Bert and GPT-2 models), MixPipe improves the training throughput by up to 2.39× over the state-of-the-art synchronous pipeline approaches.**

***Index Terms*—pipeline parallelism, data parallelism, deep learning, large models, distributed training**

I. Introduction

Deep learning is constantly making major breakthroughs by scaling the size of the Deep Neural Network (DNN) model. Large-scale models, represented by Transformers [1], have shown great promise in many domains, such as natural language processing and computer vision [2], [3]. Training the advanced Transformers with billions of parameters is challenging [4]. Due to the limited memory of modern accelerators, large models are too big to fit on a single accelerator. Even if memory optimization techniques can theoretically reduce the memory footprint during training (recomputation [5] or swap out/in strategy between GPU and host memory [6]), the huge computation operations can lead to intolerable long training times (e.g., training GPT-3 would require 288 years with a single V100 NVIDIA GPU [4]). As a result, these growing large models are typically trained in parallel on distributed accelerators (e.g., GPUs).

There are generally two kinds of parallelism for training large DNN models: data parallelism [7] and model parallelism [8]. Data parallelism reduces training time by sharding training data across multiple accelerators, but cannot handle large models individually since each accelerator requires a replica of the entire model. Model parallelism distributes a model across many accelerators to reduce the memory usage per accelerator. DNN models with layered architecture can be distributed in two ways: (1) tensor parallelism [4] splits in tensors of individual layers, and (2) pipeline parallelism [9] splits in layers of the model. The former performs all-to-all (*a2a*) communication between multiple workers, while the latter performs point-to-point (*p2p*) communication between two pipeline stages. Leveraging the intrinsic structure of Transformer based large models, the intermediate activations transferred by *p2p* communication has less overhead. Besides, in pipeline parallelism, a mini-batch of training samples is split into multiple smaller micro-batches that are executed sequentially in each worker. Since multiple devices work in parallel not only on different micro-batches but also on different pipeline stages, the *p2p* communication overhead can be further overlapped by the computation in the devices. Moreover, tensor parallelism and pipeline parallelism can be viewed as orthogonal and complementary measures for distributing models. Therefore, recent proposals tend to optimize pipeline parallelism and further combine data parallelism for efficient training of large models [10]–[13]. This work also follows this path.

However, achieving the ideal training throughput for pipeline parallelism is not trivial for two reasons. First, the training of a DNN is a sequential execution process with one forward pass followed by one backward pass [14], which causes device idle times (also called bubbles) and decreases pipeline utilization. Pipeline parallelism with synchronous (*sync*) updates requires to flush the pipelines periodically at the end of each iteration to guarantee model convergence, which leads to more bubbles. Pipeline parallelism with asynchronous (*async*) updates (e.g., PipeDream [10] and PipeDreem-2BW [11]) can reduce bubbles to almost zero by delaying weight updates, but at the cost of sacrificing convergence and thus is out of the scope of this paper. Second, the backpropagation algorithm requires each accelerator to store massive intermediate activations for backward pass, so that this memory consumption may result in low device utilization. Furthermore, because the micro-batch size is limited to the minimum available memory of all accelerators, unbalanced memory usage among pipeline stages also results in insufficient use of device computing resources [12], [13].

Recently, *sync* pipeline parallelism has made progress in reducing the number of bubbles through designing ingenious scheduling algorithms, but such solutions usually have limited training throughput due to their high peak activation memory footprints. Specifically, for some recently proposed *sync* pipeline approaches such as DAPPLE [12] and Chimera [13], they use one-forward-one-backward (1F1B) schedule that executes backward passes as early as possible, and thus, improve pipeline utilization by increasing the number of micro-batches. However, the accelerator with a pipeline of depth D has to store up to D such intermediate activations, leading to reduced computational efficiency since the micro-batch size has to be chosen to fit each accelerator in the pipeline. This under-utilization can be remedied by the measure that reduces the number of micro-batches injected into devices at the beginning (i.e., before the first backward pass, denoted by K) of the pipeline [13] . A neat example is GEMS [15], which achieves high device utilization by injecting only one micro-batch at the beginning of the pipeline. However, it introduces a lot of bubbles in turn. To sum up, smaller K improves device utilization by fully using the compute resources, but degrades pipeline utilization due to more pipeline bubbles.

In this paper, we propose a novel *sync* pipeline approach, called *MixPipe*, that achieves high overall throughput by trading off pipeline utilization and device utilization. Technically, it adopts the bidirectional pipeline architecture with higher pipeline efficiency. Inspired

*The corresponding author is Biyu Zhou.

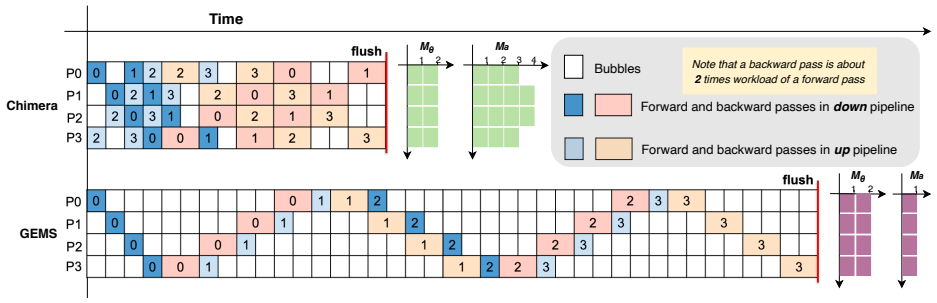


Fig. 1. *Sync* bidirectional pipeline approaches, with four pipeline stages ($D = 4$) and four micro-batches ($N = 4$) within a training iteration.

by the fact that a reasonable K plays a crucial role in the training throughput of pipeline parallelism, it contains the characteristics of multiple existing pipeline approaches by flexibly adjusting the value of K. Moreover, *MixPipe* features a novel schedule (two-forward-one-backward, 2F1B) for balanced memory usage and combines this schedule with memory-efficient 1F1B to achieve better performance. The contributions of *MixPipe* are summarized as follows:

- We deeply analyze the factors that affect the training throughput in pipeline parallelism, and propose a flexible bidirectional pipeline approach that achieves a better balance between pipeline and device utilization by adjusting the value of K.
- We design a mixed scheduling that combines 1F1B and 2F1B to achieve a more balanced memory footprint and further reduce the bubble ratio.
- We build an accurate performance model to guide the configuration of integrating our bidirectional pipeline parallelism with data parallelism. Moreover, we feature a novel device placement strategy to alleviate the *a2a* communication overhead introduced by all-reduce operations of bidirectional pipelines.
- Experiments show that *MixPipe* can improve the end-to-end performance by up to 2.39× per iteration for Transformer based language models (i.e., Bert and GPT-2 models) compared to the state-of-the-art *sync* pipeline approaches.

II. Background and Related Work

A. Basic Parallelism

Data parallelism. In data parallelism, each mini-batch is sharded evenly across multiple devices, and each device has a replica of the full model [7]. Thus, pure data parallelism cannot be used for those large models that can not fit in a single device.

Model parallelism. Model parallelism is a solution to train large models by distributing the model architecture among multiple devices in two ways: tensor parallelism and pipeline parallelism, respectively. Since the intra-layer splitting of tensor parallelism will cause expensive *a2a* communications, pipeline parallelism is more extensively studied [10]–[13]. Besides, recent work [10]–[13] also shows improved performance when combining pipeline parallelism with data parallelism.

B. Bidirectional Pipeline Parallelism

Recently, bidirectional pipeline parallelism has been proposed as an effective approach for improving training throughput. As shown in Fig. 1, it combines two pipelines in different directions (we call them down and up pipelines, respectively) together. The down and up pipelines are executed in different workers concurrently, and thus, the pipeline can be better utilized than vanilla pipeline parallelism with a single pipeline. However, existing bidirectional pipeline approaches focus on improving either pipeline utilization or device utilization, and hence cannot achieve the best performance, as detailed below. Note that we consider the typical workloads of large models, i.e., a backward pass is about 2 times workload of a forward pass.

Pipeline utilization. As shown in Fig. 1, to utilize pipeline, Chimera injects D micro-batches at once (i.e., D/2 at the beginning of down or up pipeline), where D denotes the number of pipeline stages (or depth). Due to the fully-packed bidirectional pipelines, it incurs D/2-1 bubbles (one bubble is the execution time of forward and backward passes for a micro-batch), which is about 50% reduction compared to vanilla pipeline approaches (e.g., GPipe [9] and DAPPLE [12]). We define the bubble ratio as device *idle* time (i.e., bubbles) divided by the overall training time of the pipeline. As presented in Table I, the bubble ratio of Chimera is $\frac{D-2}{3D-2}$. GEMS [15] is mainly designed for handling enormous memory requirements. The bubble ratio of GEMS is much higher than Chimera, because it executes at most two micro-batches at the same time (in Fig. 1).

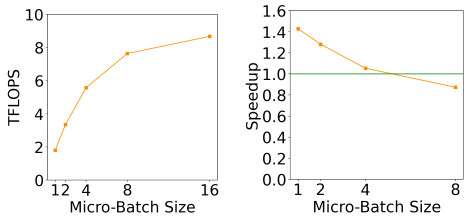
TABLE I
Comparison between different bidirectional pipeline approaches in synchronous training.

Pipeline Approaches	K	Bubble Ratio	Parameters Memory	Activations Memory
Chimera [13]	D	$\approx \frac{D-2}{3D-2}$	$2M_\theta$	$[(D/2+1)M_a,$ $D * M_a]^1$
GEMS [15]	1	$\approx \frac{D-1}{D+1/2}$	$2M_\theta$	M_a

[1] Ranges for the values across pipeline stages.

Device utilization. A larger micro-batch size (B) improves device utilization, while B is limited by the memory footprint, which mainly comes from the parameters and the intermediate activations. Table I also presents the memory usage of two bidirectional pipeline approaches, where M_θ is the memory footprint of the parameters in one worker for one stage replica, and M_a is the memory footprint of the activations in one worker for one micro-batch. Both in Chimera and GEMS, each worker maintains the parameters of two pipeline stages since there are two pipelines in two directions. We define K as the total number of micro-batches injected into the first stage of each pipeline before the first backward pass in bidirectional pipelines. Since Chimera injects D micro-batches at the beginning of bidirectional pipelines (i.e., K=D), the peak activations memory footprint is proportional to D, which leads to reduced device utilization. Although the activations memory usage can be reduced using activation recomputation technique [5], this is at a cost of about 1/3 more computation overhead [11], [13]. GEMS only injects one micro-batch at the beginning of the pipeline (i.e., K=1), benefiting from the low memory requirements from activations, GEMS can improve device utilization by setting a larger B. By analyzing the memory usage on each worker of Chimera and GEMS in Fig. 1, we

can observe that bidirectional pipeline parallelism has an extra benefit of a more balanced activation memory footprint, compared with vanilla pipeline parallelism using 1F1B schedule (e.g., in DAPPLE [12], the first worker of a pipeline of depth D has to store D such activations while the last worker requires memory for one).



(a) *FLOPS*(B) for Bert-24. (b) Throughput speed up.

Fig. 2. The throughput speed up as restricting K from D to $D/2$.

C. Motivation

We infer that, for the most advanced bidirectional pipeline parallelism solution Chimera, the overall throughput of a training iteration T is proportional to $FLOPS(B)/(D+N_{bubbles})$ (see Sec. III-A for details). $FLOPS(B)$ is the computational performance model of an available accelerator, and represents the floating point operations per second of the accelerator for the model with B. We build $FLOPS(B)$ of V100 GPU for the forward pass of a single block of Bert-24 with different micro-batch sizes, as shown in Fig. 2(a). B is the micro-batch size, and its value is inversely proportional to K. $N_{bubbles}$ is the number of bubbles in one iteration, and its value is negatively correlated with K. Fig. 2(b) shows a comparison of throughput improvements by simply reducing K from D to $D/2$. In most cases, a significant increase in throughput is observed. Besides, the gain of increasing device utilization is more significant for a smaller B (which is common for larger models due to the memory bottleneck). This phenomenon inspires us to seek a fair trade-off between pipeline utilization and device utilization through the subtle determination of K to achieve improved overall training throughput.

III. The *MixPipe* Approach

A. Flexible Bidirectional Pipelines

Overview. *MixPipe* is a bidirectional pipelining scheme. Similar with Chimera [13], we exploit the repetitive structures of large DNN models (i.e., the same block repeated multiple times), such as Bert [2] and GPT [16], which can be easily distributed into multiple computational balanced stages with an equal number of blocks. To achieve the best balance between pipeline utilization and device utilization, *MixPipe* injects moderate micro-batches into each worker at the beginning of each pipeline by flexibly regulating K. Its early forward scheduling further reduces the bubbles and ensures balanced memory usage among workers by prioritizing partial forward passes (let Γ denote the number of advanced forward passes). The optimal values of K and Γ in each worker are determined by maximizing the training throughput of one iteration (T).

Fig. 3 presents three bidirectional pipeline schemes with four pipeline stages (i.e., D=4). Here we assume each worker executes D micro-batches within a training iteration (i.e., N=D), which is the minimum to activate all pipeline stages. How to scale to more than D micro-batches (i.e., for $N>D$) will be discussed in Section III-C. In the down pipeline, stage0∼stage3 are mapped to worker P0∼P3 linearly, while the stages in the up pipeline are mapped in a dramatically opposite order. Each pipeline schedules $N/2$ micro-batches (assuming N is an even number) using our mixed schedule strategy (i.e., early forward scheduling). Here we only consider symmetric bidirectional pipelines, that is, the workload scheduling of the forward and backward passes in the two pipelines is symmetric (see Fig. 3), and leave the exploration of asymmetric bidirectional pipelines as an interesting future work.

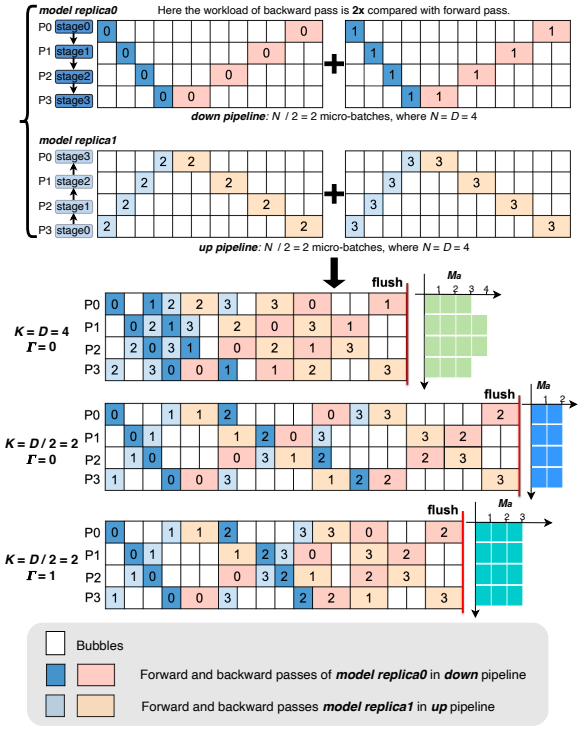


Fig. 3. The scheduling of *MixPipe* with different values of K and Γ.

The regulating of K. For bidirectional pipelines, K is the total number of micro-batches injected into the first stage of each pipeline before the first backward pass. When classic 1F1B is adopted, *MixPipe* requires to stash up to K such activations (i.e, $K \cdot M_a$). Since high activation memory usage will limit B and reduce the device utilization, it is possible to increase device utilization by decreasing K. However, restricting K introduces pipeline bubbles. As shown in Fig. 3, for two schemes with Γ=0 (i.e., only use 1F1B schedule), the peak memory usage of the scheme with K=4 is two times higher than the scheme with K=2. Consistent with our analysis, the former has fewer bubbles than the latter.

Early forward scheduling. Since bidirectional pipelines of *MixPipe* is symmetrical, K is decreased by two at a time, which decreases two activations memory usage ($2M_a$) and increases two bubbles. To achieve a more fine-grained bubbles increment, we propose an early forward scheduling (called 2F1B schedule), which advances one forward pass of a micro-batch in down (up) pipeline ahead of the previous backward pass of a micro-batch in up (down) pipeline as executing a such schedule. The number of advanced forward passes in P-(D/2-1) or P-(D/2) (both workers have peak activations memory usage) is defined as Γ. As shown in Fig. 3, for the scheme with K=2 and Γ=1, the forward pass of micro-batch3 in P1 is advanced ahead of the backward pass of micro-batch0 in P1, and thus, the number of bubbles is reduced by one while compared to the scheme with K=2 and Γ=0. In contrast, every time using 2F1B schedule will increase one such activations memory usage (i.e., M_a). Therefore, we take 2F1B schedule as an auxiliary solution of adjusting K.

We use 1F1B schedule for the micro-batches injected into the pipeline at the beginning, since there is no forward pass can be

advanced. After the first backward pass, for the workers P-(*D*/2-1) and P-(*D*/2), we cautiously use 2F1B schedule to avoid low device utilization. While for other workers, we use 2F1B schedule as much as possible until achieving balanced activations memory usage among stages, and thus, it can reduce bubbles without increasing peak memory footprint, especially in deeper pipelines.

Throughput Optimization. Now we seek for the optimal values for K and Γ to achieve the optimal throughput goal. We first give a formal expression for throughput T:

$$T = \frac{\hat{B}}{(D + N_{bubbles}) * t_{stage}}, \tag{1}$$

where $\hat{B}$ is the mini-batch size (=B∗N), and the computation time of one pipeline stage (t_{stage}) with B is computed based on a device computational performance model *FLOPS*(*FLOPs*(*B*)) (as in [17])

$$t_{stage} = \frac{L}{D} * \frac{FLOPs(B)}{FLOPS(FLOPs(B))}, \tag{2}$$

where L is the number of layers of the model, and *FLOPs*(*B*) represents the number of floating point operations of a single Transformer layer for B. It is modeled by the profiling results with different base micro-batch sizes B_{base}

$$FLOPs(B) = \frac{B}{B_{base}} * FLOPs(B_{base}), \tag{3}$$

$N_{bubbles}$ in equation (1) denotes the number of bubbles, which equals to $(5D-3K-4)/3-\Gamma$. Therefore, $(D+N_{bubbles})*t_{stage}$ is the total computation time in one iteration. To achieve the best performance, we use the largest per-accelerator micro-batch size $B_{largest}$, since *MixPipe* has a constant bubble ratio for a given D and larger B usually improves the device utilization. $B_{largest}$ can be easily computed as

$$B_{largest} = \frac{(M_{budget} - 2M_\theta) * B_{base}}{(K + \Gamma) * M_a(B_{base})}, \tag{4}$$

where M_{budget} is the memory budget of the device, and $M_a(B_{base})$ is modeled similar with *FLOPs*(B_{base}). Since $\hat{B}$ equals to $B_{largest} * D$, we finally imply that $T \propto FLOPS(B_{largest})/(D + N_{bubbles})$. As can be observed, the sum of K and Γ decreases, bubbles increases, until at some point the reduced pipeline utilization cannot be compensated by the increased device utilization.

Now, the problem of maximizing throughput can be formalized as

$$\begin{aligned} & \max T \\ s.t.\quad & (1) - (4), \\ & K = 2k, \forall k \in N \wedge k \in [1, D/2], \\ & \Gamma \in N \wedge \Gamma \in [1, (D-K)/2]. \end{aligned} \tag{5}$$

Since D is usually an integer less than 64, the optimal solution (K_{opt}, Γ_{opt}) can be obtained by exhausting the search space.

B. Hybrid Parallelism Based on Performance Modeling

MixPipe uses hybrid parallelism that integrates pipeline and data parallelism to more efficiently train large models. The bidirectional pipelines of *MixPipe* with D stages are equally replicated W times to scale to P ($P = W \cdot D$) accelerators, since we consider the large models which can be evenly distributed into multiple stages for load balance. As discussed in [4], [11]–[13], as D increases (W decreases), pipeline bubbles become more, until at some point the increased device *idle* time cannot be compensated by the reduced *a2a* communication overhead of data parallelism. Therefore, it is necessary to determine the optimal hybrid parallel configuration (i.e., the optimal values of W and D) to achieve the best performance.

Before that, we propose a novel device assignment strategy to improve the communication efficiency. *MixPipe* uses *p2p* communication in forward and backward passes, but expensive *a2a* communication (i.e., *all-reduce*) for gradients *sync*. Similar with Chimera, we use eager gradient synchronization [13] to overlap the *all-reduce* overhead in hybrid parallelism. As presented in Fig. 4, gradients *sync* ($S_{03} \sim S_{33}$) are advanced in the example of *MixPipe* with W=2 and D=4. However, Chimera ignores the influence of device assignment on communication overhead of bidirectional pipelines, which use *all-reduce* to synchronize gradients cross stage replicas (e.g., same stages in model replica 0 and 1 in Fig. 4) before the next iteration. Fig. 5(a) shows an example of device assignment of Chimera with W=2 and D=4 in two servers (each one has 4× GPUs), where *all-reduce* operations of data parallelism use high-speed NVLink, but bidirectional pipelines still use slow Ethernet bandwidth for heavy gradients *sync*. To remedy this issue, *MixPipe* tends to place all replicas of a stage (both in data parallelism and bidirectional pipeline parallelism) into the same server. As shown in Fig 5(b), all *a2a* communications in *MixPipe* use high intra-server connections to speed up gradients *sync*.

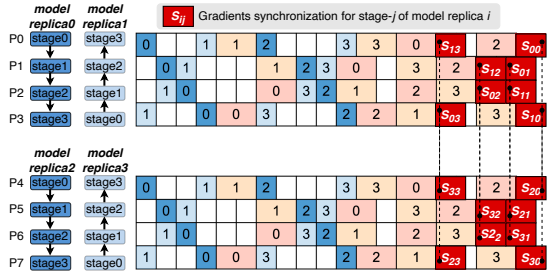
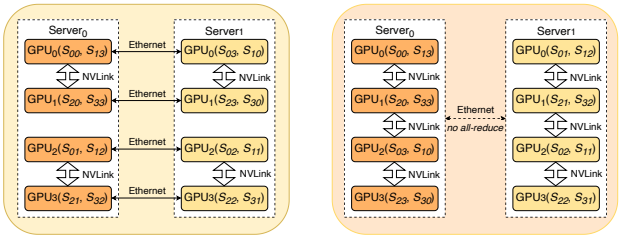


Fig. 4. Hybrid parallelism combining pipeline and data parallelism in *MixPipe* ($W = 2$, $D = 4$, $K = 1$, $\Gamma = 1$).



(a) Chimera's device assignment (b) *MixPipe*'s device assignment

Fig. 5. Device assignment customized for bidirectional pipelines. S_{ij} denotes stage-*j* of model replica *i*, and a double arrow represents an *all-reduce*.

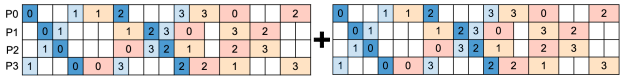
To select the best configuration of W and D, we build a performance model to predict T of hybrid parallelism for each possible configuration

$$T = \hat{B}/(t_{comp} + t_{comm}^{p2p} + t_{comm}^{a2a}), \tag{6}$$

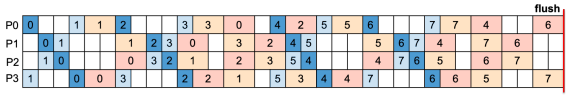
where $\hat{B} = B * N * W$. The computation time t_{comp} and the communcation overhead (t_{comm}^{p2p} and t_{comm}^{a2a}) can be modeled using similar expressions in [11] and [13], which will not be further discussed for space constraints.

C. Scale to More Micro-Batches

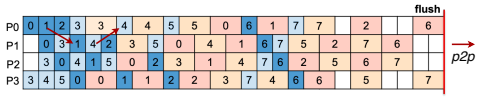
For a large $\hat{B}$, the number of micro-batches in one iteration for each worker may be more than D (i.e., N>D), especially when the storage resources are limited. To scale to a large $\hat{B}$, we first use the schedule of D micro-batches in bidirectional pipelines in Sec. III-A as a basic scheduling unit. Fig. 6(a) shows an example with N=2D micro-batches in one iteration, which has two basic units (i.e., H=2).



(a) N=2D micro-batches, where $D = 4$



(b) *Direct concatenation* (intermediate bubbles)



(c) Concatenation with K *maximizing*, i.e., K=2(D-1), (no intermediate bubbles)

Fig. 6. Scale to more than D micro-batches within a training iteration.

The simplistic scaling method is to directly concatenate H (H=N/D and N=$\hat{B}$/(W· B)) such basic units as presented in Fig. 6(b). The first forward pass of the second basic unit in each worker is advanced to occupy the bubble at the end of the first basic unit. However, this method leads to many intermediate bubbles when $N \gg D$, since the backward pass of a large model usually has about two times workload of the forward pass.

To remove the intermediate bubbles of *direct concatenation*, we propose *K maximizing* method (shown in Fig. 6(c)), which maximizes the value of K (i.e., K=2(D-1) due to $N \gg D$). Then, we schedule the first backward pass in each worker as soon as possible, so that peak activations memory can be maintained at (3D/2-1)M_a, which is lower than Chimera's scaling method (i.e., *forward doubling* [13] with 2D·M_a), and thus has better device utilization than Chimera. But the memory usage of *K maximizing* still is higher than *direct concatenation*, so we use recomputation when *K maximizing*'s memory usage exceeds the memory budget. Note that *K maximizing* has total D/2-1 bubbles, which is about a 50% reduction compared with vanilla pipeline approaches such as GPipe and DAPPLE. For N>2D, we use the schedule of the last two micro-batches as a basic unit for *K maximizing* instead of 2D micro-batches in *direct concatenation* and *forward doubling*, and concatenate (H/2-1)·D+1 basic units. Therefore, *K maximizing* not only has more space to overlap *p2p* communication in the forward passes than 1F1B schedule (in Fig. 6(c)), but also has fewer bubbles than *forward doubling* [13] in Chimera if H is odd.

IV. Evaluation

A. Experimental Setup

Hardware Configurations. We conduct our experiments on the cluster with four 8×V100 servers, where servers are connected by Ethernet and GPUs in a server are interconnected via NVLink. Each V100 GPU has 32 GB global memory.

Baselines and Implementation. We compare *MixPipe* to four *sync* approaches including GPipe, GEMS, DAPPLE, and Chimera. To be fair, all approaches are implemented in PyTorch with NCCL distributed backend. For comparing with Chimera, we reduce the search space of K to $\{2, D/2, 2(D\text{-}1)\}$ and keep the value of Γ same as Chimera, since this approach can be viewed as a sub-optimal scheme of our method (i.e., K=D and Γ=0).

Benchmark Models and Training Setup. We evaluate *MixPipe* on large transformer-based language models given in Table II, which are extensively used for NLP applications. We use Wikitext dataset for model training, where data preprocessing is the same as Pipedream-2BW [11]. The max sequence length of all models are set to 512. The sequence length and all mini-batch sizes are consistent with common practices in the machine learning community [11]–[13]. Similar with DAPPLE [12] and Chimera [13], we mainly compare the memory footprint and the training throughput, since *MixPipe* is a *sync* parallel approach without compromising convergence accuracy.

TABLE II
Benchmark models used for evaluation.

Model	# of layers	# of parameters	Mini-batch size
Bert-48	48	669933442	≥ 128
GPT-72	72	1010477056	≥ 64

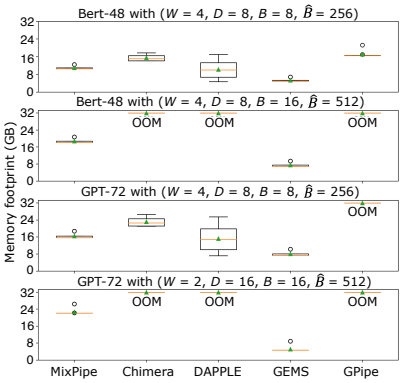


Fig. 7. Memory footprint distribution among 32 GPU devices.

B. Memory Footprint

Fig. 7 shows the memory footprint distribution comparison among 32 GPU devices in different configurations. Since GPipe injects all micro-batches at the beginning, the memory cost of this method is the highest and easily to be *OOM* (out of memory) in three out of four configurations due to the high activations memory. Chimera has the second highest memory footprint because two versions of weights and up to D micro-batches' activations have to be stashed. DAPPLE has a little lower peak memory footprint than Chimera since it maintains one replica of the model and the same amount of activations as Chimera. However, 1F1B schedule of DAPPLE leads to unbalanced memory footprints among pipeline stages, which also constrains the computational efficiency. In contrast, the mixed schedule of bidirectional pipelines in *MixPipe* determines that it has a more balanced memory footprint than DAPPLE. Furthermore, *MixPipe* has a lower peak memory footprint than DAPPLE and Chimera for all configurations in Fig. 7, since it injects fewer micro-batches (i.e., K=D/2) into workers at the beginning. While the memory footprint of GEMS is the lowest among all approaches and is also balanced, this dramatically reduces the training throughput (discussed later). Overall, *MixPipe* not only achieves a lower peak memory footprint compared with the state-of-the-art, but also has a balanced memory usage among the workers.

C. Throughput

We first compare the best performance of all approaches in the test of weak scaling with $N = D$, and then evaluate the training throughput when there are a large number of micro-batches within one iteration(i.e., $N \gg D$) for scaling to large mini-batch size $\hat{B}$.

*1) **Comparison in Weak Scaling**:* Fig. 8 and 9 present the results of weak scaling for Bert-48 and GPT-72, respectively. For all approaches, we present the best performance of the best configuration (grid searching for baselines and cost model for *MixPipe*) at different scales. The best configuration used by each approach is annotated in the legends of the figures.

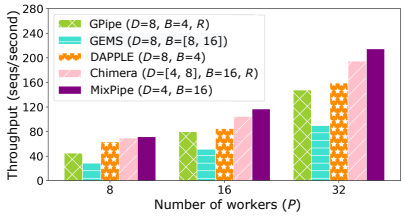


Fig. 8. Weak scaling for Bert-48. As P (i.e., the number of workers) scales from 8 to 32, $\hat{B}$ scales from 128 to 512.

For both Bert-48 and GPT-72, *MixPipe* outperforms all the baselines at all scales. For Bert-48 on the cluster with 32 GPUs, *MixPipe* outperforms GPipe, GEMS, DAPPLE, and Chmera by 1.46×, 2.39×, 1.34×, and 1.16×, respectively. GEMS has lower throughput than other methods, since it has the highest bubble ratio. To achieve the best performance, GPipe and DAPPLE use B=4 to improve the pipeline utilization but at the expense of lower device utilization. Although Chimera has fewer bubbles and uses B=16 for higher device utilization, it requires to leverage recomputation to avoid *OOM* which brings extra computation overhead. In contrast, *MixPipe* with B=16 can be fit into the device without recomputation, and therefore outperforms Chimera.

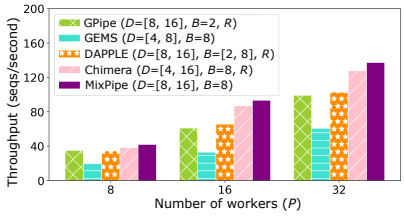
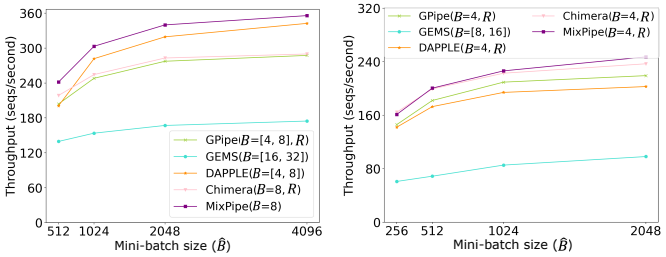


Fig. 9. Weak scaling for GPT-72. As P scales from 8 to 32, $\hat{B}$ scales from 64 to 256.

For GPT-72 on 32 GPU devices, *MixPipe* outperforms GPipe, GEMS, DAPPLE, and Chimera by 1.38×, 2.25×, 1.33×, and 1.12×, respectively. Compared to other approaches, *MixPipe* has a better tradeoff between pipeline and device utilization. *MixPipe* outperforms GPipe, DAPPLE, and Chimera mainly because no recomputation is required, and outperforms GEMS due to fewer bubbles.



(a) Bert-48. D=4 for all approaches. (b) GPT-72. D=8 for all approaches.

Fig. 10. Scale to large $\hat{B}$ on the cluster with 32 GPUs.

*2) **Evaluation for Large Mini-Batch Size**:* Fig. 10(a) presents the throughput comparison on 32 GPUs for Bert-48 when scaling to large $\hat{B}$. To achieve the best performance, GPipe and DAPPLE switch from B=4 to B=8 when $\hat{B}\geq$1024 for higher device utilization, in which case (i.e., $N\gg D$) the bubble problem is alleviated due to the large N. GEMS always uses the available largest B for higher device utilization, in which B equals to 16 for $\hat{B}$=512 and 32 for $\hat{B}\geq$1024. As shown in Fig. 10(a), the throughputs of GPipe and Chimera are lower than *MixPipe* and DAPPLE since they suffer from recomputation. *MixPipe* has a little higher throughput compared to DAPPLE because of the fewer bubbles. Overall, *MixPipe* achieves on average 1.22×, 2.01×, 1.09×, and 1.18× speedup over GPipe, GEMS (suffering from extremely high bubble ratio), DAPPLE, and Chimera, respectively.

Fig. 10(b) presents the throughput comparison on 32 GPUs for GPT-72 when scaling to large $\hat{B}$. For GPT-72, GPipe outperforms DAPPLE when $N\gg D$, this is because both approaches require recomputation but the schedule of GPipe is better to overlap the *p2p* communication. Chimera has the nearly same performance with *MixPipe* in GPT-72, since both approaches use recomputation to reduce memory usage and have the same bubbles. Benefiting from the sophisticated schedule of MixPipe with K *maximizing* which has fewer bubbles and more communication overlap as discussed in Sec. III-C, our approach achieves on average 1.10×, 2.65×, and 1.16× speedup over GPipe, GEMS, and DAPPLE, respectively.

V. Conclusion

In this work, we propose *MixPipe*, an efficient bidirectional pipeline parallelism for training large models. *MixPipe* brings new insights for achieving the best balance between pipeline and device utilization. Specifically, it uses a mixed schedule to assist the technique that adjusts the number of micro-batches injected into the pipelines at the beginning. Compared with the state-of-the-art synchronous pipeline approaches, *MixPipe* significantly improves the training throughput for Transformer-based language models.

References

[1] A. Vaswani et al., "Attention is all you need," in *NIPS*, 2017.
[2] J. Devlin et al., "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv*, 2018.
[3] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv*, 2020.
[4] D. Narayanan et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *SC*, 2021.
[5] T. Chen et al., "Training deep nets with sublinear memory cost," *arXiv*, 2016.
[6] J. Ren et al., "Zero-offload: Democratizing billion-scale model training," in *USENIX ATC*, 2021.
[7] M. Li et al., "Communication efficient distributed machine learning with the parameter server," in *NIPS*, 2014.
[8] S. Lee et al., "On model parallelization and scheduling strategies for distributed machine learning," in *NIPS*, 2014.
[9] Y. Huang et al., "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *NIPS*, 2019.
[10] D. Narayanan et al., "Pipedream: generalized pipeline parallelism for dnn training," in *SOSP*, 2019.
[11] D. Narayanan and A. Phanishayee et al., "Memory-efficient pipeline-parallel dnn training," in *PMLR*, 2021.
[12] S. Fan et al., "Dapple: A pipelined data parallel approach for training large models," in *PPoPP*, 2021.
[13] S. Li et al., "Chimera: efficiently training large-scale neural networks with bidirectional pipelines," in *SC*, 2021.
[14] M. Rhu et al., "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *MICRO*, 2016.
[15] A. Jain et al., "Gems: Gpu-enabled memory-aware model-parallelism system for distributed dnn training," in *SC*, 2020.
[16] A. Radford et al., "Language models are unsupervised multitask learners," in *OpenAI blog*, 2019.
[17] J. Guo et al., "Accudnn: A gpu memory efficient accelerator for training ultra-deep neural networks," in *ICCD*, 2019.