Seq1F1B: Efficient Sequence-Level Pipeline Parallelism for Large Language Model Training

Ao Sun[1,*] **Weilin Zhao**[2,*] **Xu Han**[2,†]
Cheng Yang[1,†] **Zhiyuan Liu**[2] **Chuan Shi**[1] **Maosong Sun**[2]
[1] Beijing University of Posts and Telecommunications, Beijing, China.
[2] NLP Group, DCST, IAI, BNRIST, Tsinghua University, Beijing, China.
{maydomine, yangcheng}@bupt.edu.cn
zwl23@mails.tsinghua.edu.cn
hanxu2022@tsinghua.edu.cn

Abstract

The emergence of large language models (LLMs) relies heavily on distributed training strategies, among which pipeline parallelism plays a crucial role. As LLMs' training sequence length extends to 32k or even 128k, the current pipeline parallel methods face severe bottlenecks, including high memory footprints and substantial pipeline bubbles, greatly hindering model scalability and training throughput. To enhance memory efficiency and training throughput, in this work, we introduce an efficient sequence-level one-forward-one-backward (1F1B) pipeline scheduling method tailored for training LLMs on long sequences named Seq1F1B. Seq1F1B decomposes batch-level schedulable units into finer sequence-level units, reducing bubble size and memory footprint. Considering that Seq1F1B may produce slight extra bubbles if sequences are split evenly, we design a computation-wise strategy to partition input sequences and mitigate this side effect. Compared to competitive pipeline baseline methods such as Megatron 1F1B pipeline parallelism, our method achieves higher training throughput with less memory footprint. Notably, Seq1F1B efficiently trains a LLM with 30B parameters on sequences up to 64k using 64 NVIDIA A100 GPUs without recomputation strategies, a feat unachievable with existing methods. Our source code is based on Megatron-LM, and now is avaiable at: `https://github.com/MayDomine/Seq1F1B.git`.

1 Introduction

In recent years, there has been a growing interest in large language models (LLMs), which have revolutionized various tasks in natural language processing [1, 2, 3, 4]. Efficient distributed training strategies [5, 6, 7, 8, 9] play a crucial role in training these large models. Among these strategies, pipeline parallelism [8, 10, 11, 12, 13] stands out due to its low communication bandwidth requirements and high scalability when integrated with other distributed training strategies.

Fundamentally, pipeline parallelism involves partitioning the model into multiple stages, with each computing device responsible for processing a stage consisting of consecutive layers. This setup inherently leads to "bubbles"—the idle time caused by the dependencies between the computation of sharded layers. Several scheduling strategies, such as GPipe [10], are proposed to address this problem, significantly reducing pipeline bubbles by splitting each mini-batch of training examples into several micro-batches. These strategies come at the expense of increased memory usage, as each stage must store all hidden states of the micro-batches generated during the forward passes until the backward passes are completed. Based on GPipe, TeraPipe [13] further splits each micro-batch along

[1]* Indicates equal contribution.
[2]† Indicates corresponding author.

the sequence dimension into micro-sequences to further reduce pipeline bubbles, but still suffer from the high memory demand for storing the hidden states of micro-batches.

To address the issue of high memory demand, one-forward-one-backward (1F1B) scheduling strategies are proposed [14, 15, 9] to rewrite GPipe to schedule backward passes in advance and make backward passes have higher execution priority than forward passes, without affecting final results. By adopting 1F1B parallel strategies, the memory demand for storing hidden states can be significantly reduced without adding extra pipeline bubbles. Other methods such as zero-bubble-pipeline [12] and 1F1B-I (1F1B with interleaved stages) [9] seek to further reduce the bubbles of 1F1B strategies but at the cost of more memory overhead and communication cost. Generally, optimizing pipeline parallelism continues to handle trade-offs between bubble ratio and memory overhead.

On the other hand, some recent efforts [16, 2] have noticed that long-sequence training benefits LLMs in many aspects, leading to increasingly longer training contexts for LLMs. However, LLMs can not simply support training longer sequences due to the quadratic time and memory complexities of Transformer attention modules in terms of sequence length [17]. Several efforts [18, 19, 20] to build efficient attention modules have been proposed to address this issue. Even so, the challenge caused by long sequences extends beyond attention modules. In distributed training scenarios, long sequences may cause various parallel methods to fail. For pipeline parallelism, such as GPipe [10] and 1F1B[14, 15, 9, 12], these methods can only use micro-batches as the minimal scheduling units. In extreme cases, a single micro-batch consisting of long sequences can lead to memory overflow. Long sequences make the issue of high memory demand in pipeline parallelism more serious.

A straightforward approach to solving such a problem is to split the micro-batches along the sequence dimension. However, such simple modification is challenging for existing 1F1B scheduling strategies such as [12] and 1F1B-I [9] because there are computation dependencies between forward and backward passes across micro-sequences, making a direct split along the sequence dimension unfeasible.

To solve such a challenge, we introduce the Seq1F1B, an efficient sequence-level 1F1B schedule, which has higher efficiency and lower memory demands than the traditional 1F1B methods. In detail, we introduce a partially ordered scheduling queue in Seq1F1B to replace the first-in-first-out (FIFO) scheduling queue in 1F1B, rewriting the scheduling strategy to preserve the exact forward and backward semantics while providing synchronous pipeline parallelism. To further improve Seq1F1B, we propose a strategy for balancing the workload across sub-sequence computations. More specifically, we balance the sub-sequence computation by designing a solution based on floating-point operations on each sub-sequence. In this design, we addressed the imbalance of computational workload caused by the attention mechanism by splitting sequences based on computational workloads rather than simply dividing them evenly along the sequence dimension.

Sufficient experiments demonstrate that Seq1F1B significantly outperforms both the 1F1B and 1F1B-I scheduling strategies in terms of memory efficiency and training throughput for training LLMs, with the sequence length ranging from 16k to 128k and the model size ranging from 2.7B to 32B. From the experimental results, the efficiency of Seq1F1B becomes more pronounced as the sequence length increases and Seq1F1B supports efficiently training a GPT with 30B parameters on sequences up to 64k tokens using 64 NVIDIA A100 GPUs without any recomputation strategies, which is unachievable with existing pipeline parallel methods.

2 Related Work

Training large language models requires using a mixture of parallel strategies, the most important of which are data parallelism, tensor parallelism, and pipeline parallelism. Data parallelism scales training models by distributing data across multiple devices[21, 22, 23, 24], each device hosting a model replica and synchronizing gradients. Zero redundancy optimizer (ZeRO) [7, 6, 25] enhances data parallelism's memory efficiency by partitioning model parameters across devices at the cost of significant communication. Tensor parallelism[8, 5] parallelize computation by partitioning matrix multiplication. In such way, Tensor parallelism effectively enhances computation efficiency but introduces high communication costs of aggregating the results of matrix multiplication, making it commonly used within the multiple workers of a single node. Since this paper focuses on improving pipeline parallelism, we will show more details for pipeline parallelism next.

For pipeline parallelism, schedules can be broadly categorized into two main types: synchronous and asynchronous. Asynchronous schedules such as asynchronous PipeDream [14] and PipeMare [11] can achieve bubble-free but suffer from the performance degradation of final trained models because they use outdated parameters to compute gradient updates. In view of this, our work focuses on synchronous pipeline schedules, as they ensure consistent semantics across different model parallel strategies. GPipe[10, 13] and 1F1B[15, 9, 26] are the most commonly used pipeline schedules following synchronous settings. Many other works are built upon these two foundation schedules.

The original GPipe[10] simply divides a mini-batch into several micro-batches. The scheduling process of GPipe has only two phases: the forward and the backward phases. Only after all forward passes for the micro-batches within a batch are complete will the backward passes be executed. During the forward phase, the hidden states of each micro-batch are enqueued into a first-in-first-out (FIFO) queue Q. During the backward phase, these hidden states are dequeued for their corresponding backward passes. Since the backward phase happens after all hidden states are queued, GPipe exhibits an $O(M)$ memory consumption, where M represents the number of micro-batches.

Based on GPipe, other methods such as TeraPipe [13] and Chimera [27] further optimize the bubble ratio of GPipe through different techniques. TeraPipe relies on the observation of causal language modeling — the computation of a given input token only depends on its previous tokens. Specifically, TeraPipe divides GPipe's micro-batch into multiple token spans and replaces the FIFO queue with a last-in-first-out (LIFO) queue to ensure the correct computation of gradients during attention backward passes. By leveraging finer scheduling units, TeraPipe effectively reduces the bubble ratio while being more memory-efficient than GPipe. Chimera adopts bidirectional pipeline parallelism, where each computing device is responsible for the workload of multiple stages. While this approach reduces the bubble ratio, each device has to store redundant parameters (as stages are not evenly distributed across devices), leading to increased memory usage.

Different from GPipe, which performs backward passes after completing all forward passes, 1F1B [9, 15] alternates between forward and backward passes (adopting a one-forward-one-backward pattern) to keep the number of hidden states in the FIFO queue Q constant. Regardless of the number of micro-batches, 1F1B mitigates excessive memory usage. Based on 1F1B, 1F1B-I[9] schedule enlarges the number of pipeline stages, and each device is assigned multiple stages. By interleaving stages among devices, 1F1B-I reduces the bubble ratio at the cost of adding more communication operators and slightly increasing memory consumption. Zero-bubble-pipeline[12] divides the backward passes into obtaining weight and input gradients separately. Zero-bubble-pipeline[12] achieves higher pipeline efficiency by delaying weight gradient computation and using dynamic programming to optimize the schedule. Zero-bubble-pipeline approach nearly achieves zero-bubble pipeline efficiency but brings more memory footprint caused by such delayment.

3 Methodology

In this section, we detail how our method works, beginning with a preliminary overview to introduce characteristics of the 1F1B schedule and language modeling. Then, we can prove why it's feasible to schedule at the sequence dimension for micro-batches in 1F1B. Following that, We will then explain how Seq1F1B works in detail and how to meet the exact semantics of original language modeling.

Building on this, we will discuss how different sequence-splitting strategies impact the scheduling order in pipeline parallelism, and we will construct an optimal solution based on the theoretical, computational load to address the load-balancing issues associated with sequence-splitting strategies, thereby enhancing the efficiency of our method.

3.1 Preliminary

1F1B includes three phases during one iteration: warm-up, steady, and cooling-down phase. Assume a 1F1B scheduling scenario where we have P workers, each responsible for one pipeline stage, such that the size of pipeline parallelism is P and the number of micro-batches is M. Each worker denoted as i, executes a forward pass during the warm-up phase. The number of warm-up micro-batches for each worker is determined by the Eq. 1.

$$\mathrm{w}_i = \begin{cases} P - i - 1 & \text{if } M > P \\ M & \text{if } M \leq P \end{cases} \tag{1}$$

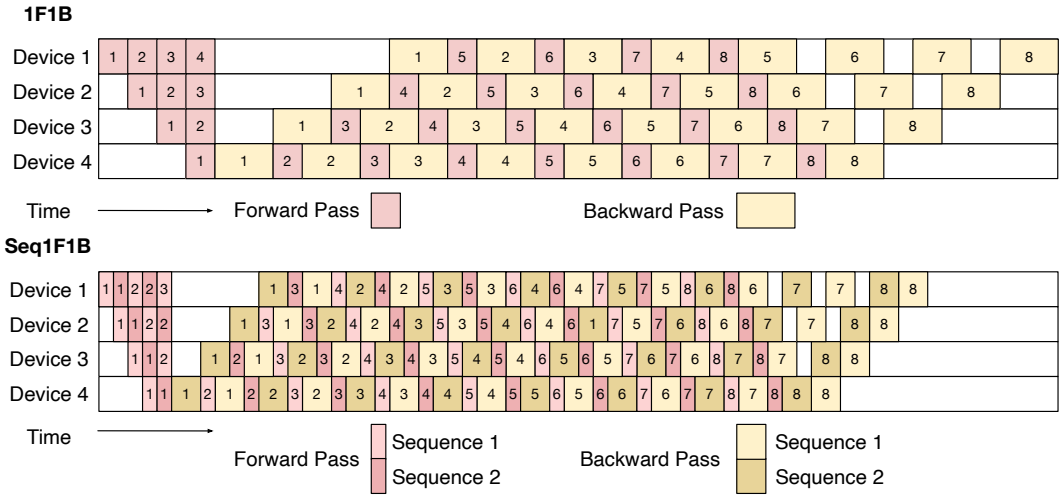

Figure 1: Execution timeline for the 1F1B and Seq1F1B schedules. Blank spaces represent idle time, also known as bubbles. The top figure illustrates the original 1F1B schedule, where each micro-batch is labeled with an ID. The bottom figure illustrates our Seq1F1B schedule, where the input is split into two sequences for better illustration. In Seq1F1B's illustration, light-colored areas represent the first sequence, while dark-colored areas represent the second sequence. Notice that the forward pass for the dark-colored sequence follows the light-colored sequence, whereas, for the backward pass, the dark-colored sequence precedes the light-colored sequence.

When w_i equals M, 1F1B degrades to the behavior of GPipe. Otherwise, during the warm-up phase, a worker responsible for an earlier stage performs one more forward pass than a worker for a subsequent stage. Each forward pass results in a hidden state that is enqueued in a FIFO queue Q to be used later for gradient computation during the backward pass. In the steady phase, each worker performs one forward pass and enqueues the resulting hidden state into Q. Following each forward pass, a hidden state is dequeued from Q and immediately to perform a backward pass for gradient computation, which is where the "one-forward-one-backward" (1F1B) name comes from. It is noted that the bubble ratio is minimal during the steady phase, and the number of one-forward-one-backward passes in this phase is given by: $M - \mathrm{w}_i$. Thus, as M increases, the proportion of the steady phase increases, which reduces the bubble ratio. After the steady phase, the 1F1B scheduling enters the cooling-down phase, which is symmetric to the warm-up phase and involves performing the same number of backward passes as in the warm-up.

The primary optimization of 1F1B is to ensure that the memory consumption of the hidden states is independent of M. The peak memory consumption for the hidden states is determined by the number of items in the queue Q at the end of the warm-up phase, where each worker holds w_i hidden states. Assuming the total memory consumption of all hidden states is A, the peak memory consumption of worker i is $w_i \frac{A}{P}$. During the steady and cooling-down phases, this consumption does not increase.

Language modeling is the most common unsupervised objective in training language models. In Language modeling's objective, each token is predicted sequentially while conditioned on the preceding tokens, embodying the principles of sequential generation, as formulated in Eq. 2.

$$P(\mathbf{x}) = \prod_{t=1}^{T} P(x_t \mid x_1, x_2, \ldots, x_{t-1}) \tag{2}$$

In the context of language modeling using Transformers, the unidirectional attention mechanism ensures that each token in a sequence can only see its predecessors, including itself.

Given a sequence of tokens $x_0, x_1, \ldots, x_n$, the output of the attention mechanism for each token can be computed as follows. Each token t_i is associated with a query vector q_i, a key vector k_i, and a value vector v_i, which will be used for attention computation. The output for each token t_i, denoted as O_i, is computed by attending over all previous tokens up to t_i, as formulated in Eq. 3.

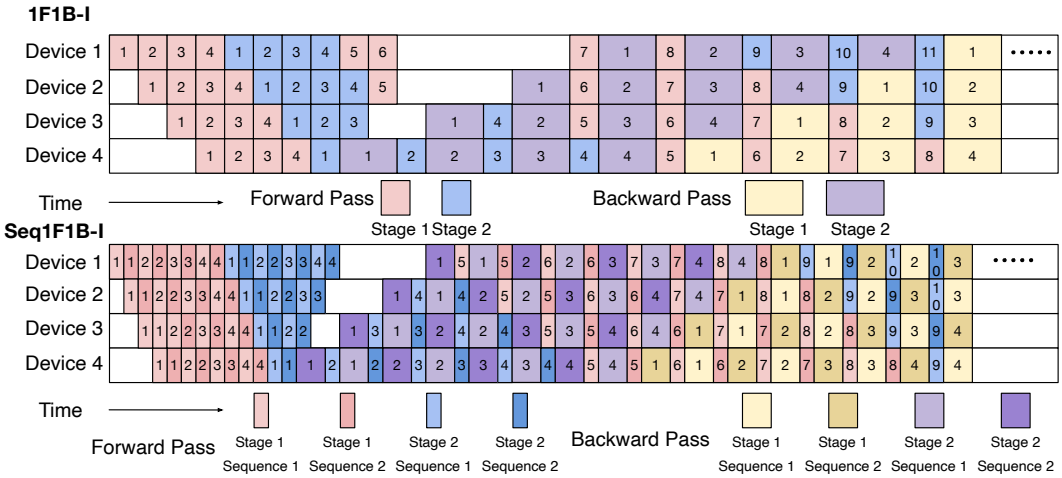

Figure 2: Execution timeline for the 1F1B-I and Seq1F1B-I schedules. The top figure illustrates the 1F1B-I schedule, where each micro-batch is labeled with an ID, and different colors distinguish the forward/backward passes of different stages. The lower part of the figure shows the Seq1F1B-I schedule, where the input is split into two segments. In Seq1F1B-I, the light-colored areas represent the first sequence and the dark-colored areas represent the second sequence.

$$O_i = \text{softmax}\left(\frac{q_i \cdot [k_0, \ldots, k_i]^T}{\sqrt{d_k}}\right)[v_0, \ldots, v_i] \tag{3}$$

Based on these characteristics, it becomes clear that to partition Transformer computation across the sequence dimension, the attention mechanism must retain the key and value vectors of all preceding tokens. The forward and backward passes also need to maintain a specific order. The forward computation of each token must follow the completion of its predecessor's computation, while the backward pass requires the subsequent token's gradients to complete its computation.

3.2 Seq1F1B

From the illustration 1, we observe that the original 1F1B schedule cannot accommodate the splitting of micro-batches along the sequence dimension because the last stage needs to immediately execute a backward pass after forwarding a micro-batch. A straightforward adaptation method is to divide each original 1F1B micro-batch into k segments and then execute a kFkB pipeline [13]. Although this schedule can reduce some bubbles in 1F1B, it does not save memory usage.

To achieve a more efficient sequence-level 1F1B pipeline schedule, we propose Seq1F1B, which is a handcrafted 1F1B schedule for sequence-level input. Specifically, Seq1F1B partitions the model into consecutive sets of layers and assigns each worker with the corresponding set (a.k.a pipeline stages). Then, Seq1F1B initializes the schedule part. Similar to 1F1B, the schedule is divided into three phases: warm-up, steady, and cooling-down.

$$\mathrm{w}_i = \begin{cases} P - i - 2 + k & \text{if } M > P \\ M & \text{if } M \leq P \end{cases} \tag{4}$$

During the warm-up phase, the number of warm-up micro-batches of each worker i is calculated according to Eq. 4, in which k represents the number of splits in the sequence. This equation ensures that the last stage can perform a backward pass on the last sequence segment of the first batch when entering the steady phase, and the worker responsible for each stage performs one more forward pass than the worker responsible for the subsequent stage. Here, we construct a partially ordered queue Q_s, where each pop returns the tail sequence from the earliest batch that has enqueued. This satisfies the first-in-first-out principle in the batch dimension and the first-in-last-out principle in the sequence dimension. In each iteration of the warm-up phase, workers execute one forward pass and enqueue the corresponding hidden states onto Q_s.

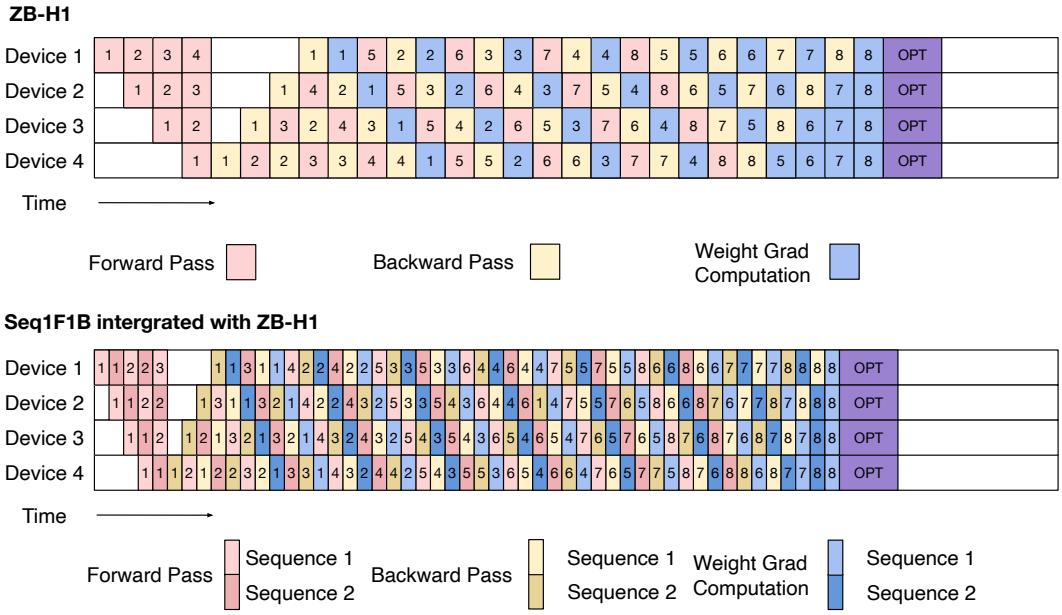

Figure 3: Execution timeline for the zero-bubble-pipeline's ZBH1 and Seq1F1B schedule intergrated with zero-bubble-pipeline' ZBH1. Where each micro-batch is labeled with an ID and different colors distinguish the forward/backward/weight computation of different stages. OPT stands for optimizer step.

In the steady phase, after each worker completes a forward pass, it dequeues from Q_s and performs a backward pass on the dequeued hidden states, following the standard 1F1B process, except that the units for forward and backward passes become a sequence segment.

In the cooling-down phase, workers dequeue the remaining warm-up hidden states from Q_s and perform backward passes sequentially.

From the timeline shown in Figure 1, it is evident that the Seq1F1B schedule offers shorter execution time and significantly fewer bubbles compared to the original 1F1B schedule. Meanwhile, it can be clearly seen that each worker now has less memory consumption since the micro-sequence is smaller than the micro-batch. Another observation is that optimizations similar to the zero-bubble pipeline can also be applied to Seq1F1B by delaying the gradient computation associated with weights in the backward pass. We will discuss this in the following section.

3.3 Seq1F1B-I

1F1B-I [9] achieves better efficiency by modifying the 1F1B schedule to support interleaved stages among workers. In 1F1B-I, each worker is assigned multiple stages. Suppose we have P workers and V stages $\{s_1, s_2, \ldots, s_V\}$ in our pipeline, where V is a multiple of P. Each worker i will handle n stages $\{s_i, s_{i+P}, s_{i+2P}, \ldots, s_{i+(n-1)P}\}$, where $n = \frac{V}{P}$. The number of warm-up micro-batches of each worker i in 1F1B-I is given in Eq. 5.

$$w_i = (P - i - 1) \times 2 + (n - 1) \times P \tag{5}$$

After completing P iterations of forward/backward passes, each worker switches its context to the next stage it is responsible for. From the Figure 2, the above part shows a 1F1B-I pipeline with P as 4 and V as 8, in which each worker handles 2 stages. 1F1B-I's schedule reduces the bubble ratio by interleaving stages among workers. However, this interleaving slightly increases memory consumption, as the number of warm-up micro-batches w_i is greater compared to 1F1B.

Similar to 1F1B-I, Seq1F1B-I further modifies 1F1B-I to achieve sequence-level scheduling, as shown in bottom part of Figure 2. From the Figure 2, Seq1F1B-I effectively reduces pipeline bubbles and maintains less memory footprint of hidden state compared with 1F1B-I. Seq1F1B-I defines the

number of warm-up micro-batches as in Eq. 6.

$$w_i = (P - i - 1) \times 2 + (n - 1) \times P + k - 1 \tag{6}$$

in which k represents the number of splits in the sequence. By using the partially ordered queue, Seq1F1B-I maintains strict order of forward/backward pass and ensures the consistent semantics of gradient updates. From the perspective of pipeline bubbles, Seq1F1B-I outperforms both Seq1F1B and 1F1B-I. In terms of memory demands, Seq1F1B-I requires slightly more memory than Seq1F1B but significantly less than 1F1B-I.

3.4 Integration with Zero-bubble-pipeline

From the illustration3, we can see Seq1F1B can integrate with ZB1P method and further reduce bubbles while reducing memory demands by splitting sequence. Such integration outperforms simple ZB1P in both memory demands and pipeline bubbles since sequence-level pipelines naturally have fewer bubbles. Furthermore, Seq1F1B can integrate with ZB2P and ZBV methods too. Theoretically, introducing a zero-bubble-pipeline to Seq1F1B should be more efficient. Even though, such a fine-grained handcraft schedule may have performance degradation under some settings. We hope our work inspires future work to solve this problem.

3.5 Workload Balance

In this section, we detail the strategy of sequence partition and workload balance consideration. Previous works, such as [13], have discussed strategies for sequence partitioning. To achieve efficient pipeline scheduling, it is crucial that the processing times for each subsequence are approximately equal to avoid pipeline bubbles. Based on this premise, we design a computation-wise partition strategy by estimating the FLOPs of sequences and constructing a theoretical solution aiming to make the FLOPs of all subsequences as closely as possible.

For a input sequence $S = (x_1, x_2, \cdots, x_n)$, we devide it into k segments $S = [S_1, \cdots, S_k]$. Each segment having a length of n_i, where $\sum_{i=1}^{k} n_i = n$. We expect the computational amount of each segment to be roughly the same, that is

$$\text{FLOPs}(S_1) = \text{FLOPs}(S_2) = \cdots = \text{FLOPs}(S_k) = \frac{\text{FLOPs}(S)}{k}. \tag{7}$$

Specifically, we use the method proposed in [28] to estimate the FLOPs for each subsequence, as formulated in Eq. 8,

$$\text{FLOPs}(S_i) = 2\ n_i\ P + 2\ L\ n_i \left(\sum_{j=0}^{i} n_j \right) d, \forall i = 1 \ldots k; \ \ \text{FLOPs}(S) = 2\ n\ P + 2\ L\ n^2 d, \tag{8}$$

in which, L is a number of layers, d is dimension of the model, and P is the total number of parameters in the model. We have k variables in Eq. 8 and k equations in Eq. 7. Therefore, we can set up the equation to get the optimial segmentation.

4 Experiments

4.1 Experimental Settings

In experiments, we measure our methods and 1F1B and 1F1B-I under variable sequence lengths, different numbers of micro-batches, different numbers of GPUs, different pipeline parallel sizes and tensor parallel sizes. Compared methods are as follows:

- Seq1F1B: Seq1F1B with computation-wise sequence partition strategy.
- Seq1F1B-I: Seq1F1B with interleaved stages and computation-wise sequence partition strategy.
- 1F1B/1F1B-I: 1F1B and 1F1B with interleaved stages in Megatron implementation.
- Seq1F1B w/o cwp: Seq1F1B without computation-wise sequence partition strategy.
- Seq1F1B-I w/o cwp: Seq1F1B-I without computation-wise sequence partition strategy.

Table 1: Settings used in experiments for training LLMs.

Model Size	Number of Layers	Attention Heads	Hidden Size	Sequence Length	PP Size	TP Size	Number of Micro-batches
2.7B	32	32	2560	16k / 24k / 32k	8	1	32 / 64
7B	32	32	4096	32k / 64k / 128k	4	8	16 / 32
13B	40	40	5120	32k / 64k / 128k	4	8	16 / 32
30B	64	64	6144	32k / 48k / 64k	8	8	32 / 64

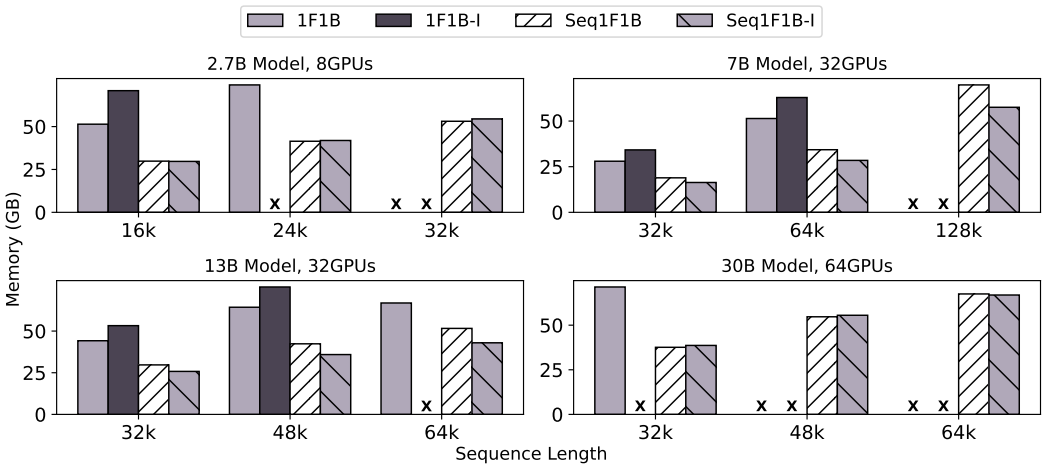

Figure 4: Peak Memory consumption of tranining a series of models under varying sequence lengths and fixed batch settings. "X" means experiments ran out of memory. We take the maximum memory consumption between all workers for better clarification.

All assessments are based on GPT model and model configuration are listed in Table 1. All experiments focus on long-sequence training since a lot of work has metioned the importance. For hyperparameter configurations, we set the number of sequence splits to four and each worker managing two stages in interleaved settings. Our implementation is based on the open-source Megatron-LM project [9] and ensures reproducibility. We adopts Megatron-V3[5]'s tensor parallelism in all experiments since it is necessary for long sequence training.

Our experiments include three cluster settings: 1) 1 node with 8 NVIDIA A100 SXM 80G GPUs interconnected by NvLink. 2) 4 nodes interconnected by a RoCE RDMA network and each node has 8 NVIDIA A100 SXM 80G GPUs interconnected by NvLink. 3) 8 nodes interconnected by a RoCE RDMA network and each node has 8 NVIDIA A100 SXM 80G GPUs interconnected by NvLink. Each measurement in the experiment is repeated 100 times, and the standard deviation is recorded.

4.2 Main Results

In Figure 4, we compared the memory consumption of our method with that of 1F1B and 1F1B-I. As can be seen, our method consistently requires less memory across all settings, and notably, it can support training a 30B model on a 64xA100 cluster, which is impossible for the traditional combination of pipeline and tensor parallelism. Additionally, we recorded TFLOPS(teraFLOPS) per GPU in our experiments to measure the hardware utilization of different methods. From the Table 2, 3, 4 and 5, our method Seq1F1B outperforms 1F1B and 1F1B-I under almost all settings in both training throughput and teraFLOPS.

However, as observed in Table 3,4,5, the Seq1F1B-I may have a performance degradation under multi-node settings. This could be due to the overly fine-grained interleaving of stage partitioning and input sequence partitioning, which also implies more communication calls in Tensor Parallelism (although the total communication volume remains unchanged), potentially leading to a decrease in performance. Another observation is that the efficiency of Seq1F1B becomes more pronounced as

Table 2: 2.7B GPT training experiments with pipeline parallel size of 8 under 8xA100 setting.

Model Size		2.7b					
Sequence Length		16384		24576		32768	
Micro-batch		16	32	16	32	16	32
Throughput (Thousands Tokens/s)	1F1B	32.0±0.0	37.1±0.0	27.0±0.0	31.4±0.0	OOM	OOM
	1F1B-I	36.4±0.0	**39.7±0.0**	OOM	OOM	OOM	OOM
	Seq1F1B	**37.3±0.0**	38.9±0.3	**32.6±0.0**	**34.2±0.0**	**28.8±0.0**	**30.1±0.2**
	Seq1F1B-I	**38.0±0.0**	38.9±0.0	**33.3±0.0**	**34.3±0.0**	**29.5±0.0**	**30.3±0.0**
TFLOPS per device	1F1B	96.9±0.0	112.3±0.0	95.5±0.1	111.1±0.1	OOM	OOM
	1F1B-I	110.3±0.1	**120.2±0.1**	OOM	OOM	OOM	OOM
	Seq1F1B	**113.1±0.0**	117.8±0.8	**115.2±0.1**	**120.9±0.1**	**116.5±0.1**	**122.0±1.0**
	Seq1F1B-I	**115.2±0.0**	118.0±0.0	**118.0±0.1**	**121.3±0.1**	**119.4±0.0**	**122.7±0.0**

Table 3: 7B GPT training experiments with pipeline parallel size of 4 and tensor parallel size of 8 under 32xA100 setting.

Model Size		7b					
Sequence Length		32768		65536		131072	
Micro-batch		8	16	8	16	8	16
Throughput (Thousands Tokens/s)	1F1B	48.2±0.1	55.3±0.2	37.3±0.0	43.1±0.0	OOM	OOM
	1F1B-I	53.0±0.3	**56.3±0.4**	41.7±0.1	44.7±0.0	OOM	OOM
	Seq1F1B	**53.5±0.3**	55.8±0.1	**43.3±0.0**	**45.0±0.1**	**30.4±0.0**	**31.6±0.0**
	Seq1F1B-I	47.2±0.9	46.2±0.8	40.9±0.4	41.0±0.3	30.0±0.0	30.4±0.0
TFLOPS per device	1F1B	99.7±0.2	114.5±0.4	107.5±0.0	124.0±0.1	OOM	OOM
	1F1B-I	109.5±0.7	**116.5±0.8**	120.0±0.2	128.7±0.1	OOM	OOM
	Seq1F1B	**110.6±0.5**	115.3±0.2	**124.6±0.1**	**129.7±0.5**	**136.7±0.1**	**142.1±0.0**
	Seq1F1B-I	97.7±1.8	95.5±1.6	117.8±1.3	118.0±0.8	135.1±0.2	136.6±0.2

Table 4: 13B GPT training experiments with pipeline parallel size of 4 and tensor parallel size of 8 under 32xA100 setting.

Model Size		13b					
Sequence Length		32768		49152		65536	
Micro-batch		8	16	8	16	8	16
Throughput (Thousands Tokens/s)	1F1B	28.9±0.1	33.4±0.1	25.3±0.1	29.3±0.1	22.6±0.1	30.0±0.0
	1F1B-I	32.2±0.2	**34.4±0.1**	28.2±0.2	30.6±0.1	OOM	OOM
	Seq1F1B	**32.9±0.1**	34.3±0.1	**29.5±0.1**	**30.8±0.0**	**26.7±0.0**	**27.8±0.0**
	Seq1F1B-I	29.7±0.4	29.8±0.3	28.0±0.2	28.3±0.1	26.4±0.1	26.8±0.1
TFLOPS per device	1F1B	106.7±0.2	123.0±0.5	109.5±0.5	126.2±0.6	111.9±0.5	135.1±0.2
	1F1B-I	118.6±0.6	**126.9±0.4**	121.9±0.7	132.2±0.4	OOM	OOM
	Seq1F1B	**121.2±0.2**	126.6±0.3	**127.3±0.4**	**133.1±0.2**	**132.5±0.0**	**137.9±0.0**
	Seq1F1B-I	109.7±1.4	110.0±1.1	121.0±1.1	122.1±0.4	130.6±0.3	132.8±0.3

the sequence length increases. This is because the computation time for each micro-sequence extends with longer sequences, thereby enhancing the benefits derived from sequence partitioning.

4.3 Ablation Results

We also conducted all experiments using Seq1F1B without computation-wise partitioning (Seq1F1B w/o cwp) and Seq1F1B-I without computation-wise partitioning (Seq1F1B-I w/o cwp) to evaluate the effectiveness of our computation-wise partition strategy. Under identical settings, employing the computation-wise partition strategy leads to performance enhancements ranging from approximately 10-30% for Seq1F1B compared to simply splitting the sequence.

Table 5: 30B GPT training experiments with pipeline parallel size of 8 and tensor parallel size of 8 under 64xA100 setting.

Model Size		30b					
Sequence Length		32768		49152		65536	
Micro-batch		8	16	8	16	8	16
Throughput (Thousands Tokens/s)	1F1B	26.4±0.1	31.2±0.2	OOM	OOM	OOM	OOM
	1F1B-I	OOM	OOM	OOM	OOM	OOM	OOM
	Seq1F1B	**31.3±0.1**	**33.1±0.2**	**28.2±0.1**	**29.6±0.1**	**25.5±0.0**	**26.8±0.0**
	Seq1F1B-I	28.0±0.4	28.4±0.2	26.5±0.2	27.1±0.2	24.8±0.1	25.2±0.1
TFLOPS per device	1F1B	104.8±0.3	123.9±0.7	OOM	OOM	OOM	OOM
	1F1B-I	OOM	OOM	OOM	OOM	OOM	OOM
	Seq1F1B	**124.5±0.2**	**131.5±0.6**	**129.4±0.3**	**135.6±0.3**	**132.6±0.0**	**139.2±0.0**
	Seq1F1B-I	111.1±1.6	113.0±1.0	121.5±1.1	124.2±0.8	128.6±0.3	130.9±0.6

Table 6: The Ablation experiments based on 2.7B GPT of sequence partitioning strategies, where "w/o cwp" indicates the absence of a computation-wise partitioning strategy.

Method	TFLOPS/device	SpeedUp
Seq1F1B w/o cwp	94.8±0.1	-
Seq1F1B	122.0±1.0	**1.28x**
Seq1F1B-I w/o cwp	103.5±0.1	-
Seq1F1B-I	122.7±0.0	**1.18x**

Across all experimental scales, Seq1F1B consistently surpassed Seq1F1B w/o cwp in performance. Table 6 highlights the ablation performance for a 2.7B model with a sequence length of 32k, demonstrating a performance boost of approximately 28% due to the computation-wise partitioning.

5 Conclusion

In this paper, we present Seq1F1B, an efficient 1F1B pipeline parallel scheduling method orienting to training Transformer-based LLMs on long sequences by decomposing the batch-level schedulable units used by typical 1F1B methods into more fine-grained sequence-level units. To achieve a better workload balance of the sequence-level pipeline, we design a computation-wise sequence partition strategy to partition the sequences well. Meanwhile Seq1F1B can integrate with other pipeline parallel methods such as 1F1B with interleaved stage or zero-bubble-pipeline. Our evaluations demonstrate that Seq1F1B outperforms the 1F1B and 1F1B-I scheduling strategies regarding memory efficiency and training throughput under variable sequence lengths and model sizes. Moreover, Seq1F1B can support the efficient training of a 30B GPT model on sequences up to 64k in length using 64xA100 GPUs, without using recomputation strategies, which is unachievable with existing pipeline parallel methods. In the future, we will thoroughly combine our method with other distributed methods to achieve better LLM training acceleration. In addition, we will systematically release our code to support the community in training LLMs to process longer sequences more efficiently.

References

[1] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[2] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[3] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[4] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[5] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

[6] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of KDD*, pages 3505–3506, 2020.

[7] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of SC20*, 2020.

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

[9] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

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

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

[12] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=tuzTN0eIO5`.

[13] Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, Dawn Song, and Ion Stoica. Terapipe: Token-level pipeline parallelism for training large-scale language models. In *International Conference on Machine Learning*, pages 6543–6552. PMLR, 2021.

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

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

[16] Jacob Buckman and Carles Gelada. Compute-optimal Context Size.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of NeurIPS*, 2017.

[18] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with io-awareness. In *Proceedings of NeurIPS*, pages 16344–16359, 2022.

[19] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, and Furu Wei. LongNet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

[20] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

[21] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[22] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010.

[23] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13(12):3005–3018, 2020.

[24] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(02):49–67, 2015.

[25] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing billion-scale model training. In *Proceedings of ATC*, pages 551–564, 2021.

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

[27] Shigang Li and Torsten Hoefler. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[28] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.