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

Ao Sun[1]*, Weilin Zhao[2]*, Xu Han[2,3]†, Cheng Yang[1]†, Xinrong Zhang[2], Zhiyuan Liu[2], Chuan Shi[1], Maosong Sun[2]

[1]Beijing University of Posts and Telecommunications, Beijing, China.
[2]DCST, IAI, BNRIST, Tsinghua University, Beijing, China.
[3]Shanghai Artificial Intelligence Laboratory, Shanghai, China.
{maydomine, yangcheng}@bupt.edu.cn, zwl23@mails.tsinghua.edu.cn
han-xu@tsinghua.edu.cn

Abstract

Training large language models (LLMs) heavily relies on distributed training strategies, among which pipeline parallelism (PP) plays a crucial role. As training sequences extend to 32k or even 128k tokens, current PP methods face severe bottlenecks, including substantial pipeline bubbles and high memory footprint, greatly hindering training throughput and model scalability. This paper introduces a sequence-level one-forward-one-backward (1F1B) PP method, named Seq1F1B, tailored for training LLMs on long sequences with high training throughput and memory efficiency. Unlike typical PP methods, which adopt batch-level pipeline schedule, Seq1F1B schedules the pipeline of training LLMs at the sequence level. It uses a computational strategy to partition sequences appropriately, significantly reducing pipeline bubbles and memory footprint. Compared to competitive PP baselines such as Megatron 1F1B PP, Seq1F1B achieves 1.14× training throughput with half memory footprint. Notably, Seq1F1B trains an LLM with 30B parameters on sequences up to 64k tokens using 64×NVIDIA A100 GPUs without using recomputation strategies, a feat unachievable with existing methods. We have released our code on GitHub to facilitate further research and development in LLM training on long sequences: https://github.com/thunlp/Seq1F1B.

1 Introduction

Efficient distributed strategies (Shoeybi et al., 2019; Li et al., 2020; Narayanan et al., 2021b) play a crucial role in training large language models (LLMs), and these LLMs have revolutionized various NLP tasks in recent years (Touvron et al., 2023; Reid et al., 2024; Jiang et al., 2024; Anil et al., 2023). Among these strategies, pipeline parallelism (PP) (Huang et al., 2019; Narayanan et al., 2021b) stands out due to its low communication bandwidth requirement and great computing resource scalability, and it can be easily integrated with other strategies such as data parallelism (DP) (Li et al., 2020; Rasley et al., 2020) and tensor parallelism (TP) (Shoeybi et al., 2019; Korthikanti et al., 2023).

PP involves partitioning a model into multiple stages, with each computing device processing a stage consisting of consecutive layers. This paradigm inherently leads to "bubbles"—the idle time caused by the execution topology between the sharded layers. Several ingenious pipeline schedule strategies have been proposed to address this bubble problem. GPipe (Huang et al., 2019) reduces bubbles by splitting each batch of training sequences into micro-batches, coming at the cost of increased memory usage, as each pipeline stage must store the intermediate states of all micro-batches generated during forward passes until backward passes are completed. To address the high memory demand of GPipe, one-forward-one-backward (1F1B) methods are proposed (Harlap et al., 2018; Fan et al., 2021; Narayanan et al., 2021b). 1F1B methods make backward passes have higher execution priority than forward passes and schedule backward passes in advance without affecting final results. Owing to this, the memory demand for storing intermediate states can be reduced without adding extra bubbles. Generally, optimizing PP relies on handling the trade-off between bubble ratio and memory footprint.

Recent studies (Buckman and Gelada; Reid et al., 2024) have highlighted the advantages of long-sequence training for LLMs in various aspects. However, training LLMs on long sequences remains challenging due to the quadratic time and memory complexities with respect to the input sequence in Transformer attention modules (Vaswani et al., 2017). Some works, such as (Liu et al., 2023; Ao et al., 2024), propose parallelizing attention computation across workers in a distributed

[*] indicates equal contribution.
[†] indicates corresponding authors.

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 8998–9008
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

cluster to enable efficient training of LLMs on long sequences. In this approach, each worker must communicate activations during attention computation and synchronize the model's weights. Consequently, these approaches exhibit poor performance when communication bandwidth is limited. Compared to these methods, PP incurs significantly lower communication overhead and is thus a more suitable choice when communication bandwidth is constrained. However, long sequence data presents new challenges, such as increased memory demands and higher bubble ratios, making it difficult to achieve effective training with PP methods. For GPipe and existing 1F1B methods, whose minimal schedulable unit is micro-batch, inevitably face the memory overflow caused by just a single micro-batch, as training sequences extend to extremely long lengths. Long sequences make balancing bubble ratio and memory footprint more challenging for PP methods.

In this paper, we introduce a Sequence-Level 1F1B (Seq1F1B) PP method. This method capitalizes on the causal self-attention mechanism of LLMs to schedule pipeline stages at the sequence level. In contrast to existing 1F1B methods (Narayanan et al., 2021b; Qi et al., 2024), Seq1F1B offers significant efficiency and memory benefits. Specifically, splitting sequences into sub-sequences allows for a significant reduction in memory footprint since only the intermediate states of sub-sequences rather than micro-batches need to be retained. Scheduling the pipeline at the sequence level yields more stages and thus reduces the bubble ratio. While, the causal nature of LLMs also causes a dependency between the forward and backward passes of different sub-sequences, i.e., the forward passes of later sub-sequences rely on earlier ones, and vice versa for the backward passes of early sub-sequences, bringing the challenge for the pipeline schedule. To this end, we introduce a partially ordered queue in Seq1F1B to replace the first-in-first-out (FIFO) queue used in existing 1F1B methods and reorganize the pipeline schedule, so that we can preserve the exact execution dependencies between forward and backward passes while providing synchronous parallelism. To further improve Seq1F1B, we propose a strategy for balancing the workload across sub-sequences rather than simply splitting sequences evenly along the sequence dimension.

Sufficient experiments demonstrate that Seq1F1B significantly outperforms recent popular 1F1B methods (Narayanan et al., 2021b; Fan et al., 2021) 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. As the sequence length increases, the efficiency of Seq1F1B becomes more pronounced. Seq1F1B supports efficiently training an LLM with 30B parameters on sequences up to 64k tokens using 64×NVIDIA A100 GPUs without using recomputation strategies, which is unachievable with existing PP methods.

2 Related Work

Training LLMs requires using a mixture of parallelism strategies, the most important of which are DP, TP, and PP (Han et al., 2021). For PP, pipeline schedules can be broadly categorized into two main types: synchronous schedules and asynchronous schedules. Asynchronous schedules such as asynchronous PipeDream (Harlap et al., 2018) and PipeMare (Yang et al., 2021) can achieve bubble-free results but suffer from the performance degradation of final trained models because they use outdated parameters to compute gradient updates. As for synchronous schedules, GPipe (Huang et al., 2019; Li et al., 2021) and 1F1B (Fan et al., 2021; Narayanan et al., 2021b,a) are the most commonly used pipeline schedules following synchronous settings. They achieve much fewer bubbles as the number of micro-batch increases and guarantee mathematical equivalent to the original training process. Given this, our work focuses on improving synchronous pipeline schedules as they ensure consistent semantics across different parallelism strategies.

The original GPipe (Huang et al., 2019) simply divides a batch into several micro-batches, and its scheduling process has only two phases: the forward phase and the backward phase. The backward passes are executed only after the forward passes of all micro-batches within a batch are completed. During the forward phase, the intermediate states of each micro-batch are enqueued into a FIFO queue Q. During the backward phase, these intermediate states are dequeued for their corresponding backward passes. Since the backward phase happens after all intermediate states are queued, GPipe exhibits an $O(M)$ memory consumption, where M represents the number of micro-batches. TeraPipe (Li et al., 2021) relies on the observation of

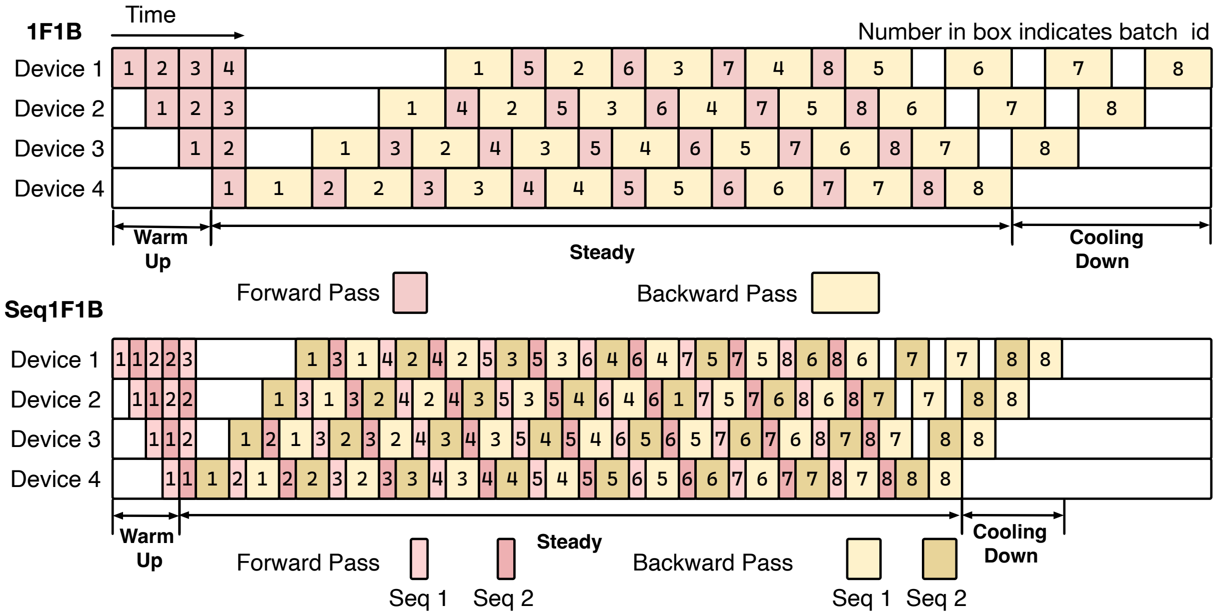

Figure 1: Execution timeline for the 1F1B and Seq1F1B schedules. Blank spaces represent idle time, i.e. bubbles. The upper figure illustrates the original 1F1B schedule, where each micro-batch is labeled with an ID and bottom interval line indicates the schedule phase of last device in the pipeline. The lower 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.

causal language modeling, where the computation of a given input token only depends on its previous tokens, 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 in backward. By using finer schedulable units (token spans), TeraPipe reduces the bubble ratio and improves memory efficiency. Chimera (Li and Hoefler, 2021) adopts a bidirectional schedule, where each device is responsible for processing multiple stages. While Chimera 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, 1F1B (Narayanan et al., 2021b; Fan et al., 2021) alternates between forward and backward passes (adopting a 1F1B pattern) to keep the number of intermediate states in the FIFO queue Q constant. Regardless of the number of micro-batches, 1F1B mitigates excessive memory usage. Based on 1F1B, 1F1B with interleaved stages (1F1B-I) (Narayanan et al., 2021b) enlarges the number of pipeline stages and assigns each device 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 (ZB1P) (Qi et al., 2024) divides the backward pass into weight gradient computation and input gradient computation separately. This approach achieves higher pipeline efficiency by delaying weight gradient computation and optimizing the schedule using dynamic programming. ZB1P nearly achieves zero-bubble pipeline efficiency but brings more memory footprint caused by delaying memory release. 1F1B methods are the most popular for training LLMs, yet still suffer from difficulties in balancing bubble ratio and memory footprint, which is the issue we want to solve.

3 Methodology

In this section, we first give a preliminary overview to introduce the characteristics of the 1F1B schedule and language modeling. Then, we prove why it is feasible to schedule the pipeline of training LLMs at the sequence level for micro-batches in 1F1B. Finally, we explain how Seq1F1B works in detail and how it meets the exact semantics of original language modeling.

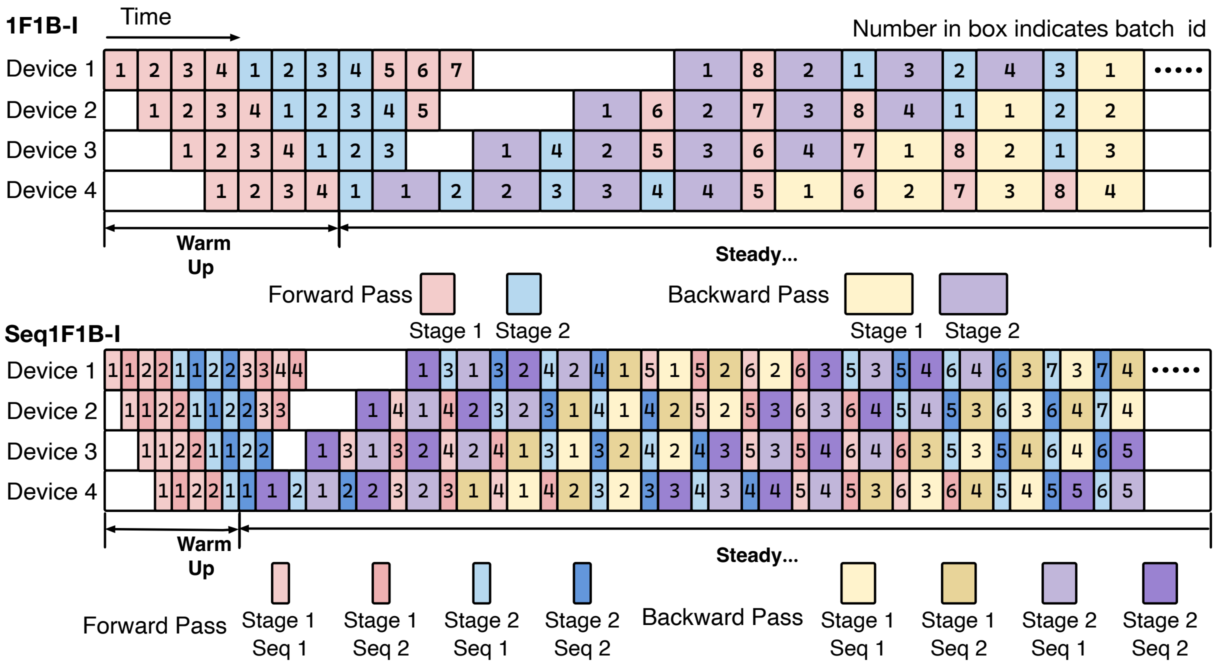

Figure 2: Execution timeline for the 1F1B-I and Seq1F1B-I. The upper figure illustrates the 1F1B-I schedule, where each micro-batch is labeled with an ID, and distinct colors represent the forward and backward passes of different stages. The lower figure shows the Seq1F1B-I schedule, where the input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

3.1 Preliminary

As shown in Figure 1, **1F1B** includes three phases to train a batch of sequences: warm-up, steady, and cooling-down phases. Given P devices (e.g., GPUs) to perform a 1F1B schedule to train M micro-batches, with each device responsible for one pipeline stage, the size of PP is P. After splitting the batch into M micro-batches, during the warm-up phase, each device executes the forward passes of the first few micro-batches, and the number of forward passes w_i executed by the i-th device is

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

When $M \leq P$, 1F1B degrades to the behavior of GPipe and does not process the steady phase. Otherwise, during the warm-up phase, a device responsible for an earlier stage performs one more forward pass than the device responsible for its subsequent stage. Each forward pass results in intermediate states enqueued in a FIFO queue Q to be used later for the gradient computation of backward passes.

During the steady phase, each device performs one forward pass and enqueues the resulting intermediate states into Q. After a device executes a forward pass, the device dequeues specific intermediate states from Q and immediately executes a backward pass for gradient computation, where the "1F1B" name comes from. Note that the bubble ratio is minimal during the steady phase, and the number of 1F1B passes in the steady phase is given by $M - \mathrm{w}_i$. As M increases, the proportion of the steady phase in the entire pipeline increases, which reduces the bubble ratio. After the steady phase, the 1F1B schedule enters the cooling-down phase, which is symmetric to the warm-up phase and involves executing the same number of backward passes as in the warm-up phase.

The primary optimization of 1F1B is to ensure that the memory consumption of intermediate states is independent of M. The peak memory consumption for intermediate states is determined by the number of items in the queue Q at the end of the warm-up phase, where each device holds w_i intermediate states. Assuming the total memory consumption of intermediate states is A, the peak memory consumption of the i-th device is $\frac{A \times w_i}{\sum_{j=1}^{P} w_j}$. During the steady and cooling-down phases, this consumption does not increase since each backward pass frees the storage space for its associated intermediate states.

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

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

where T is the sequence length. In the context of language modeling using Transformers, the causal attention mechanism ensures that each token in a sequence can only see its predecessors to process hidden states, including itself. Given a sequence of input token states $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, the output of the attention mechanism for each token can be computed as follows. Each token's state $\mathbf{x}_i$ is mapped into a query vector $\mathbf{q}_i$, a key vector $\mathbf{k}_i$, and a value vector $\mathbf{v}_i$, and output for each token $\mathbf{o}_i$ is computed by attending over all previous tokens as follows,

$$O_i = \text{softmax}\left(\frac{\mathbf{q}_i^\top \cdot [\mathbf{k}_1, \ldots, \mathbf{k}_i]}{\sqrt{d_k}}\right)[\mathbf{v}_0, \ldots, \mathbf{v}_i], \tag{3}$$

where d_k is the vector dimension. Based on these characteristics, it is clear that to partition the Transformer computation across the sequence dimension 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 pass 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. This computational dependency needs to be fully considered in the sequence-level pipeline schedule.

3.2 Framework of Seq1F1B

From Figure 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 kFbB pipeline (Li et al., 2021). 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. Similar to 1F1B, the schedule of Seq1F1B is also divided into three phases: warm-up phase, steady phase, and cooling-down phase. During the warm-up phase, the number of sub-sequences of the i-th device is computed according to

$$\mathrm{w}_i = \begin{cases} P - i - 1 + k & \text{if } M > P \\ M & \text{if } M \leq P \end{cases}, \quad i \in [1, P], \tag{4}$$

where P is the size of the PP and k indicates the number of divisions of the sequence. This equation ensures that the last stage can perform a backward pass on the last sub-sequence of the first micro-batch when entering the steady phase, and the device responsible for each stage performs one more forward pass than the device responsible for the subsequent stage. Here, we construct a partially ordered queue Q_s, where each pop returns the tail sequence from the earliest enqueued intermediate states. This satisfies the FIFO principle in the batch dimension and the first-in-last-out (FILO) principle in the sequence dimension. In each step of the warm-up phase, devices execute one forward pass and enqueue the corresponding intermediate states of sub-sequences into Q_s. During the steady phase, after each device completes a forward pass, it dequeues intermediate states from Q_s and performs a backward pass, following the standard 1F1B process, except that the units for forward and backward passes become a sub-sequence. During the cooling-down phase, devices dequeue the remaining intermediate states from Q_s, perform backward passes for remaining subsequences and accumulate the gradient to ensure mathematical equivalent with other synchronous schedules.

From the timeline shown in Figure 1, it is evident that the The Seq1F1B schedule offers a shorter execution time and significantly fewer bubbles compared to the original 1F1B schedule. Meanwhile, it can be seen that each device now has less memory consumption since the sub-sequence is smaller than the micro-batch.

3.3 Framework of Seq1F1B-I

As shown in Figure 2, 1F1B-I (Narayanan et al., 2021b) achieves better efficiency by modifying the 1F1B schedule to support interleaved stages among devices. In 1F1B-I, each device is assigned multiple stages. Suppose we have P devices and V stages $\{s_1, s_2, \ldots, s_V\}$ in our pipeline, where V is a multiple of P. The i-th device 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 device i in 1F1B-I is as follows,

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

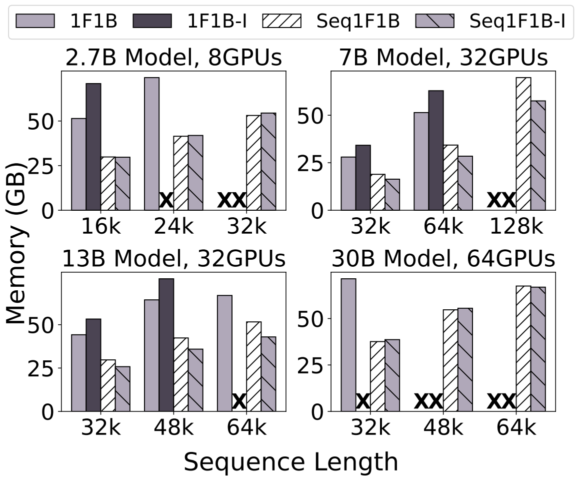

Figure 3: Peak Memory consumption of training a series of models under varying sequence lengths and fixed batch settings. "X" means experiments ran out of memory. We take the maximum memory consumption between all devices for better clarification.

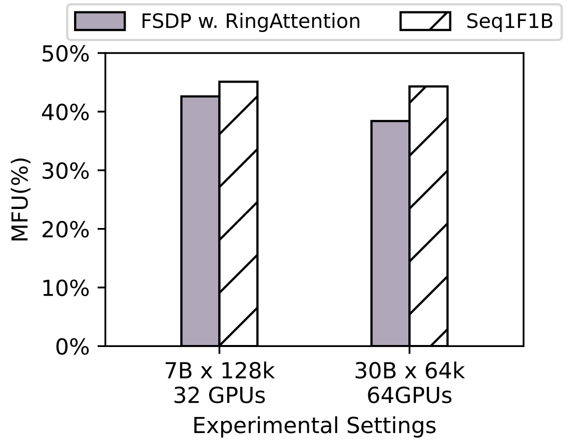

Figure 4: Comparison of model FLOPS utilization (MFU) between Seq1F1B and FSDP with RingAttention for long sequence training across various settings.

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

Similar to 1F1B-I, Seq1F1B-I further modifies 1F1B-I to achieve a sequence-level schedule. From Figure 2, Seq1F1B-I effectively reduces pipeline bubbles and the memory footprint of intermediate states compared to 1F1B-I. Seq1F1B-I defines the number of warm-up sub-sequences as

$$w_i = (P - i) \times 2 + (n - 1) \times P + k - 1, i \in [1, P], \quad (6)$$

where P is the size of the PP and k indicates the number of divisions of the sequence. Using the partially ordered queue, Seq1F1B-I maintains a strict order of forward and backward passes and ensures the consistent semantics of gradient updates. In terms of reducing pipeline bubbles, Seq1F1B-I outperforms both Seq1F1B and 1F1B-I. Besides, Seq1F1B-I requires slightly more memory than Seq1F1B but significantly less than 1F1B-I.

3.4 Workload Balance

In this section, we detail the strategy of sequence partition and workload balance consideration. Previous works, such as (Li et al., 2021), have discussed strategies for sequence partitioning. To achieve an efficient pipeline schedule, the processing cost for each sub-sequence must be balanced to minimize pipeline bubbles. To this end, 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 sub-sequences as closely as possible. For a input sequence $S = \{x_1, x_2, \cdots, x_n\}$, we devide it into k segments $S = \{S_1, S_2, \cdots, S_k\}$. Each segment has 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

$$\begin{aligned} \text{FLOPs}(S_1) &= \text{FLOPs}(S_2) \\ &= \cdots = \text{FLOPs}(S_k) \\ &= \frac{\text{FLOPs}(S)}{k}. \end{aligned} \quad (7)$$

Specifically, we use the method proposed in (Hoffmann et al., 2022) to estimate the FLOPs for each subsequence, formulated as

$$\begin{aligned} &\text{FLOPs}(S_i) = 2n_i P + 2Ln_i \left(\sum_{j=0}^{i} n_j \right) d, \forall i \in [1, k], \\ &\text{FLOPs}(S) = 2nP + 2Ln^2 d, \end{aligned} \quad (8)$$

in which, L is the number of layers, d is the 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), and thus we can set up the equation to get the optimal segmentation.

3.5 Integration with Zero-bubble-pipeline

Optimizations similar to ZB1P can also be applied to Seq1F1B by delaying the gradient computation

Model Size		2.7b					
Sequence Length		16k		24k		32k	
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 1: 2.7B GPT training experiments with PP size of 8 under 8× A100 setting.

Model Size		7b					
Sequence Length		32k		64k		128k	
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 2: 7B GPT training experiments with PP size of 4 and TP size of 8 under 32×A100 setting.

associated with weights in the backward pass. In this way, Seq1F1B can integrate with the ZB1P method and further reduce bubbles while reducing memory demands by splitting the 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 so, such a fine-grained handcraft schedule may cause performance degradation in some settings. We provide a detailed timeline of Seq1F1B integrated with Zero-bubble-pipeline in Appendix A.2. We hope our work inspires future work to solve this problem.

4 Experiments

4.1 Experimental Settings

In experiments, we measure Seq1F1B, Seq1F1B-I, 1F1B, and 1F1B-I under variable sequence lengths, different numbers of micro-batches, different numbers of GPUs, and different PP and TP sizes. Additionally, we assess the performance of Seq1F1B and Seq1F1B-I without employing the computation-wise sequence partition strategy. Furthermore, we examine the performance of FSDP with RingAttention and compare it against Seq1F1B. Compared methods are as follows:

(1) Seq1F1B: Seq1F1B with computation-wise sequence partition strategy.

(2) Seq1F1B-I: Seq1F1B with interleaved stages and computation-wise sequence partition strategy.

(3) 1F1B/1F1B-I: 1F1B and 1F1B with interleaved stages in Megatron implementation.

(4) Seq1F1B w/o cwp: Seq1F1B without computation-wise sequence partition strategy.

(5) Seq1F1B-I w/o cwp: Seq1F1B-I without computation-wise sequence partition strategy.

(6) FSDP w. RingAttention: Fully Sharded Data Parallel with RingAttention.

All assessments are based on the GPT model and focus on long-sequence training since a lot of work has mentioned its importance. For Seq1F1B and Seq1F1B-I, we set the number of sequence splits to 4, and each device manages two stages in interleaved settings. We provide detailed configurations of the hardware and other hyperparameters in Appendix A.1.

4.2 Main Results

In Figure 3, we compare the memory consumption of our method with that of 1F1B and 1F1B-I. As

Model Size		13b					
Sequence Length		32k		48k		64k	
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

Table 3: 13B GPT training experiments with PP size of 4 and TP size of 8 under 32× A100 setting.

Model Size		30b					
Sequence Length		32k		48k		64k	
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 4: 30B GPT training experiments with PP size of 8 and TP size of 8 under 64× A100 setting.

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

Table 5: The Ablation experiments are based on 2.7B GPT of sequence partitioning strategies, where "w/o cwp" indicates the absence of a computation-wise partitioning strategy.

Settings	k	Memory (GB)	Throughput(TGS)
2.7B× 32k	4	51.9	**3655.10**
	8	40.5	3449.26
	16	**39.7**	3262.12
7B× 128k	4	67.2	**987.43**
	8	59.5	968.75
	16	**56.5**	918.77
13B × 64k	4	53.5	**868.72**
	8	45.8	824.97
	16	**43.5**	750.01
30B× 64k	4	68.0	**418.71**
	8	**48.2**	387.51
	16	**48.2**	364.09

Table 6: Memory performance(GB) and throughput performance(Tokens/GPU/Second) of Seq1F1B with varying k values across the experimental configurations detailed in Table 7

can be seen, our method consistently requires less memory across all settings. Notably, it can support training a 30B model on a 64× A100 cluster, which is impossible for the traditional combination of PP and TP. Additionally, we recorded TFLOPS (teraFLOPS) per GPU in our experiments to measure the hardware utilization of different methods. From Table 1, 2, 3 and 4, our method Seq1F1B outperforms 1F1B and 1F1B-I under almost all settings in both training throughput and teraFLOPS.

However, as observed in Table 2, 3, and 4, 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 that more communication calls in TP (although the total communication volume remains unchanged) potentially leads to a decrease in performance. Another observation is that the efficiency of Seq1F1B becomes more pronounced as 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.

To better assess Seq1F1B's performance in long sequence training, we evaluated the training throughput of both Seq1F1B and FSDP w. RingAttention under two scenarios: training a 7B model with a 128k sequence length and training a 30B model with a 64k sequence length. As illustrated in Figure 4, Seq1F1B demonstrates superior training throughput compared to FSDP with RingAttention across both settings. Such performance advantage arises from Seq1F1B's significantly lower communication overhead compared to FSDP.

4.3 Ablation Results

To assess the efficiency of our computation-wise partition strategy, we 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 evenly.

Across all experimental scales, Seq1F1B consistently surpassed Seq1F1B w/o cwp in performance. Table 5 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.

Also, we evaluate Seq1F1B's performance under different k settings, and the experiment results are shown in Table 6. As can be seen, the memory consumption decreases as k increases since each device has fewer memory requirements for each sub-sequence. However, the throughput decreases as k increases from 4 to 16 because overly fine-grained sequence partitioning can lead to performance degradation due to low hardware utilization.

5 Conclusion

In this paper, we present Seq1F1B, an efficient 1F1B pipeline parallel schedule 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 that evenly distribute computational load across devices. 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 schedules 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 64× A100 GPUs without 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.

Limitations

The current implementation of Seq1F1B is optimized for long-context training in LLMs, which may result in performance degradation when dealing with short context such as 4k/8k. We recommend using Seq1F1B in environments with limited communication bandwidth, as the PP incurs fewer communication costs compared to other parallel strategies.

Acknowledgments

This work is supported by the National Key R&D Program of China (2022ZD0160502), Institute Guo Qiang at Tsinghua University.

References

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

Sun Ao, Weilin Zhao, Xu Han, Cheng Yang, Zhiyuan Liu, Chuan Shi, and Maosong Sun. 2024. Burstattention: An efficient distributed attention framework for extremely long sequences. *arXiv preprint arXiv:2403.09347*.

Jacob Buckman and Carles Gelada. Compute-optimal Context Size.

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*, pages 431–445.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open*, 2:225–250.

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

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. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

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.

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. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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

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. *Proceedings of the VLDB Endowment*, 13(12):3005–3018.

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*, pages 1–14.

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

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. TLDR: This work presents a novel approach, Ring Attention with Blockwise Transformers (Ring Attention), which leverages blockwise computation of self-attention and feedforward to distribute long sequences across multiple devices while fully overlapping the communication of key-value blocks with the computation of blockwise attention.

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

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021b. 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.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. 2024. Zero bubble (almost) pipeline parallelism. In *The Twelfth International Conference on Learning Representations*.

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

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

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*.

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

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

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*, 3:269–296.

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

Table 7: Settings used in experiments for training LLMs.

A Appendix

A.1 Hyperparameters settings and Hardware configurations

For the hyperparameter settings, we provide a detailed list of all model configurations used in our experiments, along with their corresponding hyperparameters, in Table 7. Our implementation is based on the open-source Megatron-LM project (Narayanan et al., 2021b) and ensures reproducibility. We adopt Megatron-V3 (Korthikanti et al., 2023)'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,k 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. Our method's loss curve remains the same with Megatron's 1F1B and 1F1B-I under the same model initialization setting

A.2 Seq1F1B's timeline integrated with ZB-1P

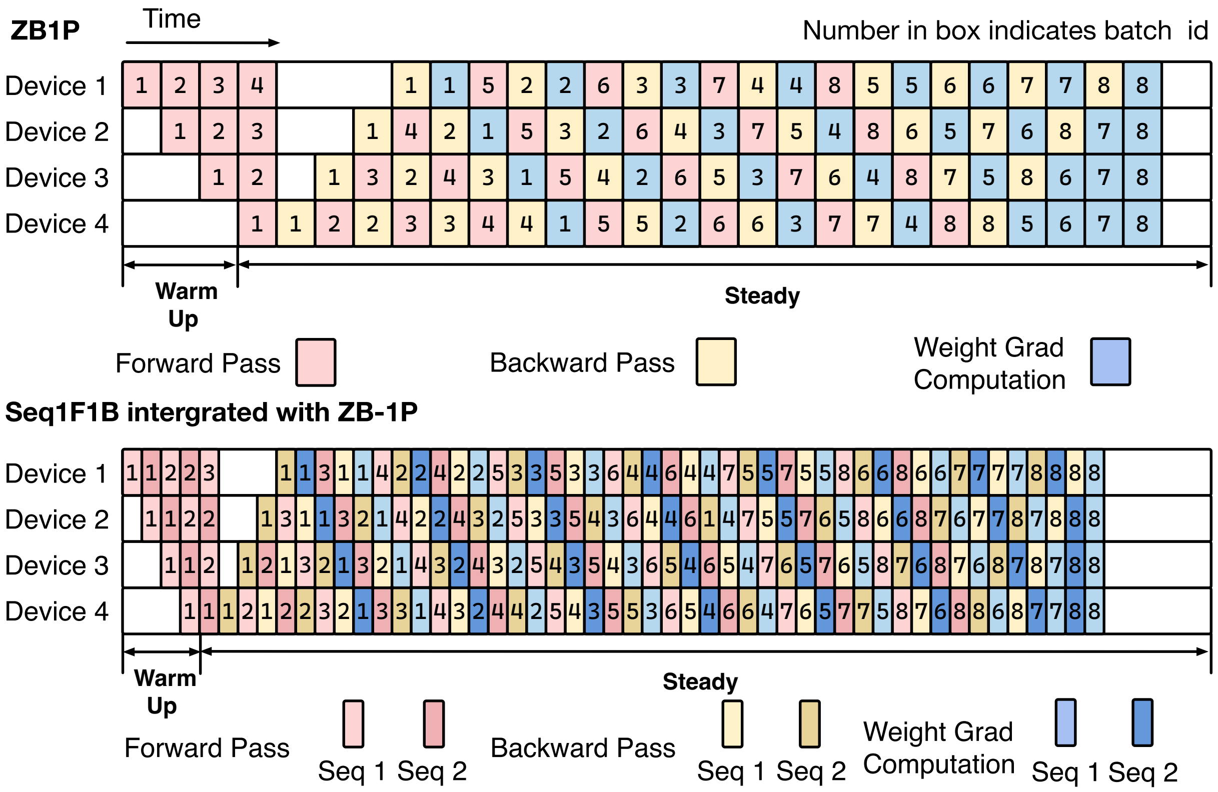

Figure 5: Execution timeline for the zero-bubble-pipeline's ZB1P and Seq1F1B schedule intergrated with zero-bubble-pipeline's ZB1P. Each micro-batch is labeled with an ID and different colors to distinguish the forward/backward/weight computation of different stages.