Enhancing Memory Efficiency in Large Language Model Training Through Chronos-aware Pipeline Parallelism

Xinyuan Lin*
linxinyu22@mails.tsinghua.edu.cn
Dept. of Electronic Engineering
Tsinghua University
Beijing, China

Chenlu Li*
cll@birentech.com
BirenTech
Shanghai, China

Zongle Huang
huangzl23@mails.tsinghua.edu.cn
Dept. of Electronic Engineering
Tsinghua University
Beijing, China

Chunyu Wang
cywang@birentech.com
BirenTech
Shanghai, China

Bo Xiao
yxun@birentech.com
BirenTech
Shanghai, China

Huazhong Yang
yanghz@tsinghua.edu.cn
Dept. of Electronic Engineering
Tsinghua University
Beijing, China

Shishi Duan
burnessduan@birentech.com
BirenTech
Shanghai, China

Yongpan Liu
ypliu@tsinghua.edu.cn
Dept. of Electronic Engineering
Tsinghua University
Beijing, China

Abstract

Larger model sizes and longer sequence lengths have empowered the Large Language Model (LLM) to achieve outstanding performance across various domains. However, this progress brings significant storage capacity challenges for LLM pretraining. High Bandwidth Memory (HBM) is expensive and requires more advanced packaging technologies for capacity expansion, creating an urgent need for memory-efficient scheduling strategies. Yet, prior pipeline parallelism schedules have primarily focused on reducing bubble overhead, often neglecting memory efficiency and lacking compatibility with other memory-efficient strategies. Consequently, these methods struggle to meet the storage demands of storage capacity for next-generation LLM.

This work presents ChronosPipe, a Chronos-aware pipeline parallelism for memory-efficient LLM pretraining. The core insight of ChronosPipe is to treat HBM as a fast but small 'cache,' optimizing and exploiting temporal locality within LLM pretraining to enhance HBM utilization. ChronosPipe introduces a pipeline scheduling strategy, Chronos-Pipe, to reduce the extrinsic overhead that disrupts the temporal locality of activations. Additionally, it leverages Chronos-Recomp and Chronos-Offload to efficiently harness the intrinsic temporal locality of activations and weights in Deep Neural Networks. Experiment results show that ChronosPipe can expand the trainable model size by 2.4x while maintaining comparable throughput, achieving 1.5x better than the 1F1B strategy combined with recomputation.

*Both authors contributed equally to this research.

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Pipeline Parallelism, Recomputation, Offload, Memory-Efficient Schedule, Temporal Locality

1 INTRODUCTION

In recent years, Large Language Models (LLMs) have demonstrated exceptional performance across diverse domains [35], leading to an urgent demand for storage capacity during pre-training. This demand can be attributed to two main reasons outlined below: **Parameter Scaling**: The scaling law [1] posits that LLMs with increased parameter counts yield superior performance. This trend is exemplified by the rapid expansion from GPT-1 (0.12B parameters) [2] to Llama 3.1 (405B parameters) [3], representing a more than 3000-fold increase in model size within a mere six-year span. **Sequence Length Extension**: Longer sequence lengths have proven instrumental in enhancing model capabilities, particularly in multimodal applications [36] that necessitate the processing of extensive video and audio sequences. Consequently, the memory requirements for activations, weights, gradients, and optimizer states keep soaring.

Contemporary GPU training systems utilize high-bandwidth memory (HBM) for runtime variable storage, but it now faces significant challenges. First, HBM integration requires advanced packaging techniques such as through-silicon via (TSV) [38] and microbump [37], resulting in substantial manufacturing costs. For instance, HBM accounts for 50-60% of the total costs in NVIDIA's H100 SXM5 module [42]. Moreover, next-generation HBM poses considerable problems. Increasing the count of stacking DRAM die within the restricted packaging height (720 um) necessitates hybrid bonding [39], whose thermal management and yield optimization [40] are extremely challenging.

Therefore, LLM training necessitates memory-efficient scheduling strategies while existing Pipeline Parallelism (PP) predominantly emphasizes performance optimization and overlooks memory constraints. Currently, most works focus on pipeline bubble reduction through fine-grained task division [4–8] or utilizing tasks in other dimensions [9, 10]. However, these methods fail to address activation storage imbalances across pipeline stages (as shown in Fig. 1(b)) and may inadvertently increase activation and weight storage requirements. Few works notice this issue and propose offloading activations to CPUs [11] or additional GPUs [12]. However, unlike the offloading of optimizer states, which only happens once per mini-batch, activation offloading is more frequently conducted across different micro-batches, thus raising a higher demand for offloading bandwidth.

(a) ***Limited HBM Capacity when Model and Seq Len Scale***

(Model Size[1], Seq Len)	(70B, 4K)	(70B, 16K)	(140B, 4K)
Peak Activation/device (GB)[2]	35.20	140.80	70.40
Model State/device (GB)[2]	19.69	19.69	39.38
Total Mem (GB)[2]	54.89	160.49	109.78
HBM Size@A100 (GB)	80		

1: model change number of layer based on LLAMA-70B 2: 1F1B evaluation at PP8_TP8,micro-batch size=2

(b) ***Temporal Locality Matters but is Ignored***

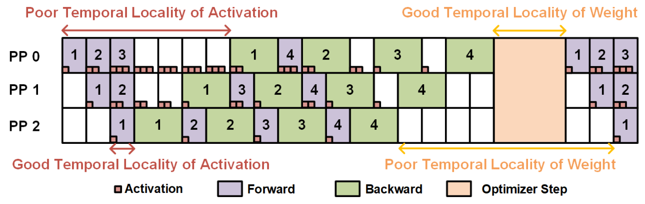

Poor Temporal Locality of Activation
Good Temporal Locality of Weight
PP 0
PP 1
PP 2
Good Temporal Locality of Activation
Poor Temporal Locality of Weight
Activation
Forward
Backward
Optimizer Step

Figure 1: (a) limited HBM capacity is difficult to meet the demands of next-generation LLM. (b) existing pipeline parallelism ignores temporal locality

Moreover, current LLM training predominantly employs hybrid parallelism, yet existing PP strategies lack optimal compatibility with other memory-efficient scheduling approaches. The ZeRO series [13] represents state-of-the-art data parallelism (DP) techniques, distributing model states across DP machines. In hybrid DP and PP configurations, distributing optimizer states (ZeRO-1) necessitates inter-DP rank communication only at **mini-batch** granularity. However, further distributing weight gradients (ZeRO-2) and weights (ZeRO-3) requires more frequent communication at **micro-batch** granularity, imposing higher bandwidth demands between DP ranks. Consequently, hybrid DP-PP implementations [41] typically utilize only ZeRO-1, leaving substantial potential for reducing weight and gradient storage. When designing PP strategies that are more compatible with ZeRO-2/3, such as Breadth First PP [14] and ZeROPP [15], these approaches reduce DP bandwidth requirements at the cost of a significant increase in activation memory, leaving no perfect solution.

Existing PP strategies lack memory efficiency due to their failure to address an intrinsic property of Deep Neural Network (DNN): **temporal locality**. In these networks, forward computation precedes backpropagation, causing activations in shallower layers to be generated early but released late, thus incurring a worse temporal locality. In pipeline parallelism, where layers are distributed across machines, the first stage that contains the shallowest layer experiences peak activation storage.

Unlike previous work, we notice the issues mentioned above and introduce the concept of temporal locality to pipeline parallelism for the first time. We propose ChronosPipe, which enables the training of models 2.4 times larger than 1F1B [5] on the same hardware with comparable throughput while maintaining compatibility with various memory-efficient scheduling strategies. The ChronosPipe solution comprises three components: Chronos-Pipe, Chronos-Recomp, and Chronos-Offload, which are described below. **Chronos-Pipe** minimizes peak activation storage by reducing micro-batch execution time, thus expediting the backward pass that consumes activations. This, in effect, optimizes activation temporal locality. **Chronos-Recomp** exploits the poorer temporal locality of shallower layer activations and discards them from HBM by selectively recomputing to achieve higher efficiency. **Chronos-Offload** takes advantage of poorer temporal locality of deeper layer weight and discards them from HBM by offloading the process of optimizer update to CPU. Since this process overlaps with PP's warmup and cooldown phase, it lowers demand on offload bandwidth and the processing speed of CPU.

Our main contributions are as follows.

- We found that temporal locality in DNN is the primary cause of imbalanced activation storage in PP. This imbalance limits the size of trained models on current devices.
- For activation memory savings, we introduce temporal locality into the PP schedule (Chronos-Pipe) and recomputation (Chronos-Recomp). Chronos-Pipe eliminates unnecessary intervals, optimizing the temporal locality of activation (Section 4.1). Chronos-Recomp discards activation with poor temporal locality from HBM by selectively recomputing shallow layers to achieve higher efficiency (Section 4.2).
- For Model State storage capacity, we show that a ZeRO-2-compatible PP could built based on Chronos-Pipe (Section 4.3). Additionally, we designed the offload strategy based on temporal locality (Chronos-Offload), which can discard model states with poor temporal locality from HBM by offloading optimizer updates of deeper layers to the CPU (Section 5.1).
- End-to-end evaluation on ChronosPipe is carried out on a cluster comprising up to 64 accelerators. Experiments show that Chronos-Pipe can expand the trainable model size by 2.4x while maintaining comparable throughput, achieving

1.5x better than the 1F1B strategy combined with recomputation.

2 BACKGROUND AND MOTIVATION

In this section, we will introduce pipeline parallelism (PP) (Section 2.1), typical recomputation strategies (Section 2.2), and offloading strategies (Section 2.3). We will then discuss the motivation for leveraging temporal locality (Section 2.4).

To facilitate clearer explanations throughout the paper, we will use the following symbols.

- m_a:memory consumption for activation in whole Neural Network(exclude input embedding layer and language modeling head)
- P:# of pipeline stages
- m:# of micro-batches executed within a training iteration
- T_{fwd}: execution time of forward pass in one micro-batch
- T_{bwd}: execution time of backward pass in one micro-batch

2.1 Pipeline parallelism

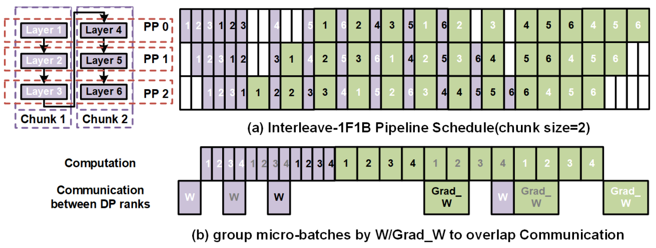

Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Layer 6
Chunk 1
Chunk 2
PP 0
PP 1
PP 2
(a) Interleave-1F1B Pipeline Schedule(chunk size=2)
Computation
Communication between DP ranks
W
Grad_ W
(b) group micro-batches by W/Grad_W to overlap Communication

Figure 2: (a) Interleave-1F1B schedule. (b) ZeRO-compatible pipeline schedule

Pipeline Parallelism (PP) divides the model into multi-layer blocks mapped across different PP stages. These stages work together to train all samples within a mini-batch, enabling mini-batch Stochastic Gradient Descent. During the forward pass, samples in each mini-batch begin computation at the first stage, with subsequent blocks executing their computations in a pipelined manner until the final forward computation completes at the last stage. In the backward pass, gradients for each mini-batch flow in the reverse direction. As a result, PP only requires a single tensor to be sent and received per block, significantly reducing communication bandwidth demands. This approach is often deployed across nodes and is a key component in hybrid parallelism.

Pipeline parallelism (PP) at the mini-batch level suffers from significant pipeline bubbles, prompting the common practice of splitting mini-batches into smaller micro-batches. By dividing the workload, the first stage, after processing the initial micro-batch, can immediately start on subsequent micro-batches rather than idling, thus reducing pipeline bubbles [4]. When the number of micro-batches increases, the classic 1F1B (one-forward-one-backward) scheduling strategy [5] is often used, as shown in Fig. 1(b). During the steady phase, a forward computation block is initiated only after a backward block finishes, avoiding the simultaneous launch of multiple forward passes in the warm-up phase, which would otherwise increase peak memory usage. However, this approach creates an imbalance in activation memory usage across PP stages. In a single micro-batch, the first stage starts forward computation early but is the last to complete backward passes and release memory. As a result, the first stage reaches peak activation memory of m_a, while the last stage's activation memory usage is only $\frac{m_a}{p}$. Additionally, Fig. 1(b) illustrates that tasks with dependencies between adjacent PP stages execute sequentially, leading to non-overlapping point-to-point (P2P) communication. Some scheduling methods address this by inserting independent task blocks from other micro-batches between dependent blocks, enabling overlapping P2P communication. This delay caused by dependency constraints is called the "interval" delay in this paper.

Blocks can be broken down into smaller tasks to reduce pipeline bubbles further, though this may increase memory requirements. The Interleaved-1F1B approach [6] divides blocks into more chunks and schedules multiple chunks launches during the warmup phase, as shown in Fig. 2(a). When the workload is divided into v chunks (assigning layer 1, $p+1, \ldots, (v-1)p+1$ to stage 0; layer 2, $p+2, \ldots, (v-1)p+2$ to stage 1, and so forth), the bubble size is reduced to $\frac{1}{v}$ of that in the traditional 1F1B, but peak activation memory increases to $m_a(1+\frac{p-1}{pv})$. The Zero Bubble approach [8] goes a step further by splitting the backward pass (BP) into two phases: activation gradient computation (BPA) and weight gradient computation (BPW). This arrangement theoretically enables a bubble-free pipeline. However, the activation gradients generated by BPA are required to be retained until BPW is completed, which raises memory demands. Additionally, splitting the BP disrupts the original reuse of activation gradients for all-gather operations between BPA and BPW, and reduces the overlap between computation and communication.

Designing a ZeRO-compatible pipeline schedule is challenging due to the trade-off between DP communication bandwidth and activation memory storage. Communication between DP ranks at the micro-batch level is required for a schedule to be compatible with ZeRO-2 or ZeRO-3. To overlap this communication, existing methods, such as Breadth-First PP [14] and ZeROPP [15], use grouped execution of micro-batches, as shown in Fig. 2(b). This approach processes multiple micro-batches simultaneously, allowing communications of weights and weight gradients can be overlapped with computations, but it also causes a substantial increase in activation memory usage. Since weights and weight gradients are stored in 16-bit and 32-bit precision, respectively, achieving ZeRO-3 PP requires at least a 1.5x increase in DP communication bandwidth or an increase in the number of micro-batches per group compared to ZeRO-2 PP. Consequently, some implementations choose to use only ZeRO-2 PP [3].

2.2 Recomputation

Recomputation allows only a subset of activations generated in the forward pass (called checkpoints) to be retained, with all other activations regenerated during the backward pass, trading off memory usage for additional computation. Previous work [16] has explored optimal checkpoint placement, given a fixed number of checkpoints, to minimize activation memory, achieving memory savings at sublinear cost for chain structure of DNN. Additionally, the benefits of recomputation vary across different operators. Recent studies suggest recomputing only the attention part in transformers [17] or recomputing some non-linear operators [11] to reduce memory usage with minimal computational overhead effectively. So far, it is still expensive to recompute the projection operator in LLM, which dominates the remaining activation memory usage. Moreover, [49] further introduced temporal locality-aware recomputation principle and demonstrated its efficacy in data parallelism. Although pipeline stages inherently exhibit temporal locality variations in PP, coarse-grained PP fails to leverage intra-block temporal locality variance for recomputation optimization within the simplistic structure of LLMs. Existing methods thus fail in single-chunk PP. Moreover, Direct application of this principle to multi-chunk PP (e.g., interleaved 1F1B scheduling) risks inducing steady-phase bubbles, resulting in significant throughput degradation.Therefore, [49] fail to solve imbalanced memory in PP.

2.3 Offloading Strategy

Offloading techniques allow tensors to be moved from memory-constrained GPUs to other GPUs or CPUs, trading memory usage for data movement. In BPipe [12], GPU offloading is used to address activation memory imbalances in pipeline parallelism. However, this approach requires high-bandwidth intra-node communication links like NVLink rather than slower inter-node links like Ethernet or InfiniBand. CPU offloading utilizes PCIe to offload activation or model states. Theoretically, CPU offloading for activation memory reduction is well-suited for long-sequence training [11]. However, limited PCIe bandwidth may leave some activation in HBM in practice. For model states, previous work, such as ZeRO-Offload [18], offloads the process of optimizer-step to the CPU, storing both optimizer states and master weights in CPU-side DRAM. To minimize GPU idle time caused by latency in the CPU's optimizer step, this approach imposes significant requirements on both offload bandwidth and CPU computational capacity. Overall, Recent studies still have high bandwidth requirements for the Offloading Strategy.

2.4 Motivation: Temporal-Locality Matters

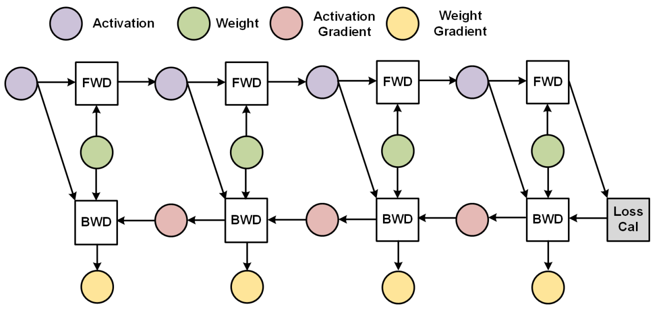


Figure 3: forward and backward pass of deep neural network

In the training of DNN, the forward pass is conducted first, followed by backpropagation, which naturally contains differences in temporal locality. As shown in Fig. 3, activations of shallow layers are generated first during the forward pass but are the last to be released in the backward pass, resulting in poor temporal locality. In contrast, weights of deep layers can be updated early during backpropagation, though they are not needed until the end of the forward pass, providing ample time for updating the optimizer state—a property that can be further leveraged.

In pipeline parallelism, temporal locality reveals some additional interesting phenomena:

Higher Peak Activation Storage Requirements: The PP stage responsible for shallow-layer activations releases them last, which increases the activation storage requirements. Additionally, intervals can exist between blocks with dependencies, and these intervals can accumulate across multiple blocks, further worsening the temporal locality for shallow layers. This is the reason that the interleaved 1F1B schedule results in a higher peak storage demand.

Skewed Distribution of Peak Activation Storage: As shown in Fig. 2(a), the peak activation storage in the interleaved 1F1B schedule tends to occur at Stage 0. When the chunk size is set to 2, the ratio of peak activation storage between shallow and deep layers is $(2p - 1) : p$, indicating an obvious bias.

Sufficient Time for Process of optimizer-step: Beyond the natural advantage of chain-structured networks providing additional time for optimizer update on deep-layer weights, increasing the PP will provide more sufficient time for optimizer-step. Large PP does not impact the execution latency of each micro-batch while the amount of model state that needs updating per stage is reduced, thereby lowering the communication and processing demands for weight updates.

3 OVERVIEW OF ChronosPipe

Inspired by the principle of efficient storage utilization in Cache, ChronosPipe applies cache's core concepts to HBM. The central idea of ChronosPipe is to optimize and leverage temporal locality during LLM pretraining, enabling more efficient utilization of the HBM.

ChronosPipe classifies the factors affecting temporal locality in LLM training into two main types. First, temporal locality varies across different layers within the same network; this influence is referred to as **intrinsic temporal overhead**, capturing the impact on temporal locality inherently introduced by the model's structure. Second, the same layer may display different temporal locality depending on the scheduling strategy used; this influence is termed **extrinsic temporal overhead**, capturing the effects on temporal locality introduced by scheduling differences.

To address extrinsic temporal overhead, ChronosPipe draws on the first critical insight from Cache: optimize locality in applications. As illustrated in Fig. 4, ChronosPipe introduces a PP scheduling strategy called Chronos-Pipe to advance the backward passes of both shallow and deep layers, minimizing the lifespan of activation. Detailed explanations of Chronos-Pipe are provided in Section 4.1. This scheduling strategy reflects ChronosPipe's approach to optimizing temporal locality in LLM training.

While scheduling can only reduce extrinsic temporal overhead, addressing intrinsic temporal overhead still requires additional strategies. To tackle this problem, ChronosPipe draws on a second critical insight from Cache: discard data with poor locality from Cache. As illustrated in Fig. 4, ChronosPipe removes data with poor intrinsic temporal locality from HBM. This involves Chronos-Recomp, a recomputation strategy for shallow-layer activations, and Chronos-Offload, an offloading strategy for deep-layer weights.

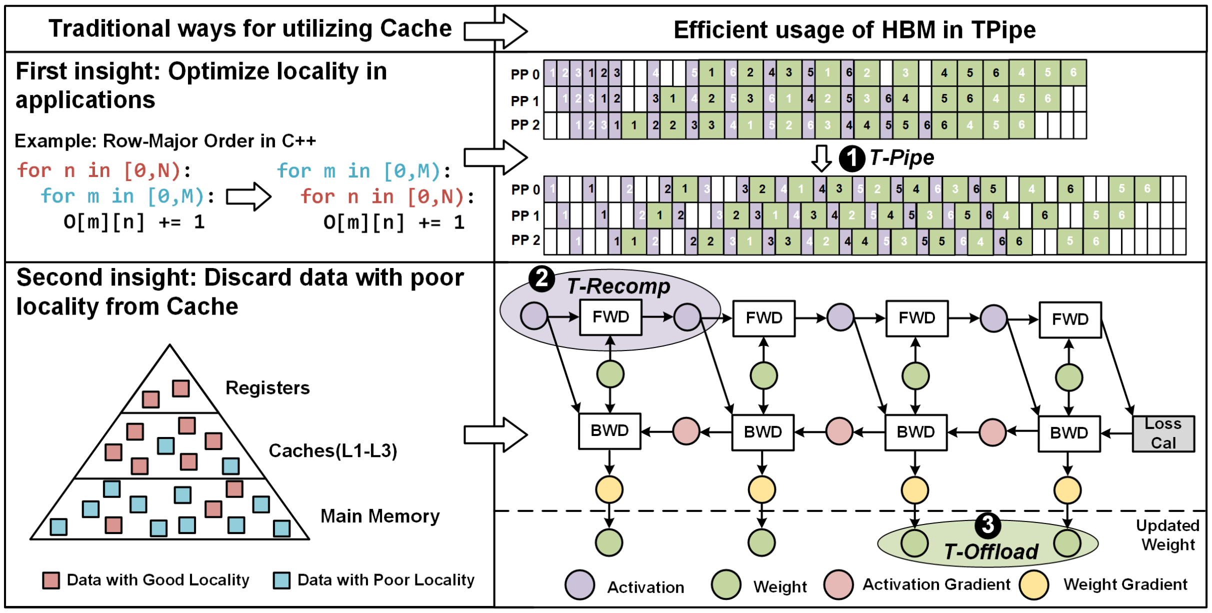
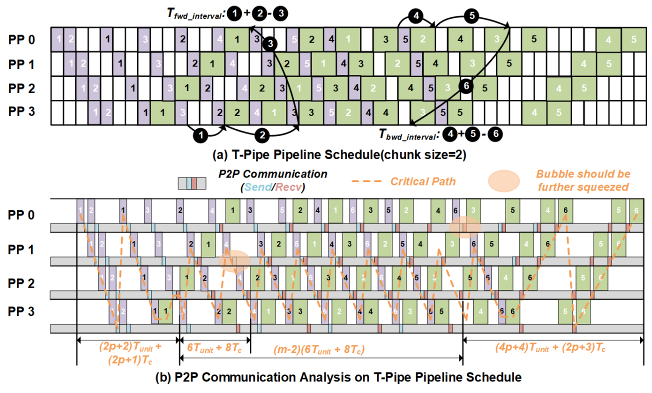


Figure 4: Overall scheduling of ChronosPipe. The core idea behind ChronosPipe is to treat HBM as a fast but limited "cache." Chronos-Pipe enhances activation temporal locality in PP scheduling while Chronos-Recomp and Chronos-Offload discard activation and weight with poor temporal locality from HBM.

Detailed explanations of Chronos-Recomp and Chronos-Offload are provided in Sections 4.2 and 5.1, respectively. Through recomputation and offloading, ChronosPipe leverages temporal locality to improve memory efficiency in LLM training.

4 Chronos-aware SCHEDULE FOR ACTIVATION MEMORY SAVINGS

4.1 Chronos-Pipe: Chronos-aware Pipeline



$T_{fwd\_interval}$: 1+2-3
$T_{bwd\_interval}$: 4+5-6
PP 0
PP 1
PP 2
PP 3
(a) T-Pipe Pipeline Schedule(chunk size=2)
P2P Communication (Send/Recv)
Critical Path
Bubble should be further squeezed
$(2p+2)T_{unit}$ + $(2p+1)T_c$
$6T_{unit} + 8T_c$
$(m-2)(6T_{unit} + 8T_c)$
$(4p+4)T_{unit} + (2p+3)T_c$
(b) P2P Communication Analysis on T-Pipe Pipeline Schedule

Figure 5: (a) Chronos-Pipe Pipeline Schedule(chunk size=2). (b) Analysis of overhead of P2P communication in Chronos-Pipe.

Chronos-Pipe Schedule. The primary goal of Chronos-Pipe is to mitigate the impact of interval accumulation in pipeline scheduling, which worsens activation temporal locality. To leverage the temporal locality differences across layers, we further split blocks into multiple chunks and prioritize completing backpropagation as early as possible. Unlike interleaved 1F1B, we aim to execute tasks with dependencies in adjacent stages as tightly as possible to minimize the negative effects of interval accumulation across multiple tasks on shallow-layer temporal locality. As shown in Fig. 5(a), with a chunk size of 2, Chronos-Pipe's scheduling introduces an interval only between the two chunks in both the forward and backward passes.

Theoretical Analysis. A further theoretical analysis of Chronos-Pipe's impact on activation storage requirements and P2P communication is provided. For simplicity, we assume $T_{bwd} = 2T_{fwd}$ and set T_c for a single P2P communication to zero unless stated otherwise. With these assumptions, basic unit of time T_{unit} in the pipeline scheduling diagram (e.g., Fig. 5(a)) is $\frac{T_{fwd}}{2p}$, and the activation storage for each task is $\frac{m_a}{2p}$.

In Chronos-Pipe, activation storage can be estimated by analyzing the activation lifespan, approximately 75% of m_a. As shown in Fig. 5(a), Stage 0 launches chunk 2 of the fourth micro-batch before releasing the activation storage from chunk 2 of the third micro-batch. Since tasks on a single stage execute in cycles of $6T_{unit}$, the accumulated activation storage for chunk 2 can be represented as $\lceil \frac{T_{life\_chunk2}}{6T_{unit}} \rceil$, where T_{life_chunk2} denotes the lifespan of chunk 2's activations, precisely the time from the completion of its forward pass to the beginning of the corresponding backward pass at PP stage 0. Similarly, the accumulation of activation storage for chunk 1 can be expressed as $\lceil \frac{T_{life\_chunk1}}{6T_{unit}} \rceil$. Additionally, we observe that the forward execution times for both chunks T_{fwd_chunk1}, T_{fwd_chunk2} are each pT_{unit}, while the backward execution times T_{bwd_chunk1}, T_{bwd_chunk2} are each $2pT_{unit}$. There is also an extra interval $T_{fwd\_interval}$ between the forward chunks, derived from steps 1, 2, and 3 in Fig. 5(a) as $(3+6\lceil \frac{(p-3)}{6} \rceil - p)T_{unit}$. Similarly, an additional interval $T_{bwd\_interval}$ exists between the backward chunks, derived from steps 4, 5, and 6 in Fig. 5(a) as $(3+6\lceil \frac{(2p-3)}{6} \rceil - 2p)T_{unit}$. Thus, for Stage 0, the lifespans of the activations for chunk 1 and chunk 2, denoted T_{life_chunk1},T_{life_chunk2} can be expressed as:

$$T_{life\_chunk2} = T_{fwd\_chunk2} + T_{bwd\_chunk2} - 2T_{unit} \quad (1)$$

$$\begin{aligned} T_{life\_chunk1} = & T_{fwd\_chunk1} + T_{fwd\_interval} + \\ & T_{life\_chunk2} + T_{bwd\_interval} + T_{bwd\_chunk1} \end{aligned} \quad (2)$$

As a result, the peak activation storage requirements for chunks 1 and 2 are $\lceil \frac{2}{3} + \lceil \frac{(p-3)}{6} \rceil + \lceil \frac{(2p-3)}{6} \rceil + \frac{p}{2} \rceil \frac{m_a}{2p}$, $\lceil \frac{(3p-2)}{6} \rceil \frac{m_a}{2p}$ respectively. When P is large, these values approach $\frac{m_a}{2}$, $\frac{m_a}{4}$, making the total peak activation storage approximately 75% of m_a.

In Chronos-Pipe, when the task is divided into v chunks, the bubble overhead from P2P communication is approximately v times that of the 1F1B schedule. The scheduling of Chronos-Pipe with two chunks under the condition $T_c \neq 0$ is considered to illustrate this. As shown in Fig. 5(b), the Chronos-Pipe execution time can be divided into three phases: T_{warmup}, T_{steady} and $T_{cooldown}$.

• The warmup phase primarily includes the forward pass of the first micro-batch, calculated as $(2p+2)T_{unit} + (2p+1)T_c$.

• The Steady phase covers the forward pass of chunk 1 for the remaining $(m-2)$ microbatches, totaling $(m-2)(6T_{unit} + 8T_c)$.

• The cooldown phase mainly involves the backward pass of the last micro-batch, calculated as $(4p+4)T_{unit} + (2p+3)T_c$.

Adding the time of these three phases together, the total time is $6(m+p-1)T_{unit} + (4p+8m-12)T_c$. Exactly, Fig. 5(b) shows that the warmup and cooldown phases can each remove one T_c of bubble overhead, leading to an adjusted total time of $6(m+p-1)T_{unit} + (4p+8(m-2)+2)T_c$. In comparison, the execution time for 1F1B is derived as $6(m+p-1)T_{unit} + (2p+4(m-2))T_c$. From this, we can see that Chronos-Pipe with two chunks has a P2P communication overhead slightly greater than twice that of 1F1B. This is because P2P communication between Stage 0 and Stage (P-1) appears in the critical path of the Chronos-Pipe schedule during the warmup and cooldown phases. This analysis holds for other values of v, where the P2P communication bubble overhead will slightly exceed v times that of 1F1B. Further analysis shows that the theoretical bubble overhead for Chronos-Pipe with two chunks is: $\frac{6(p-1)T_{unit}+[4p+8(m-2)+2]T_c}{6(p-1+m)T_{unit}+[4p+8(m-2)+2]T_c}$ whereas the bubble overhead for 1F1B is: $\frac{6(p-1)T_{unit}+[2p+4(m-2)]T_c}{6(p-1+m)T_{unit}+[2p+4(m-2)]T_c}$ .Set $T_c = 0$, the bubble overhead for Chronos-Pipe matches that of 1F1B. When $T_c = 0.05T_{unit}$, $m = 128$, $p = 4$, the theoretical bubble overheads are 8.27% for Chronos-Pipe and 5.37% for 1F1B. Due to the increased P2P communication in Chronos-Pipe, the Model Flop Utilization (MFU) is slightly reduced.

Implementation of Chronos-Pipe. In Chronos-Pipe's implementation, partial asynchronous communication is introduced to reduce bubble size. In mainstream frameworks like DeepSpeed and Megatron, the 1F1B configuration uses synchronous P2P communication. However, in Chronos-Pipe, there can be significant intervals between chunks during forward and backward passes. If synchronous P2P communication is used in these intervals, much of the time would be spent waiting for communication to complete. To address this, Chronos-Pipe modifies P2P communication between chunks to be asynchronous, thereby achieving higher throughput. As illustrated in Fig. 5(b), after Stage 0 completes the backward computation for chunk 2, it can proceed with subsequent tasks until just before Stage (P - 1) begins the corresponding backward pass for chunk 1, at which point the P2P communication is completed. Other P2P communications remain synchronous, which is consistent with mainstream frameworks.

4.2 Chronos-Recomp: Chronos-aware Recomputation

Recompute Shallow Layer First! Chronos-Pipe aims to shorten the lifespan of activations in shallow layers by reducing the interval. Yet, it cannot fully eliminate the intrinsic differences in temporal locality across layers within DNN. We can further mitigate this issue by discarding data with low temporal locality from the fast but limited HBM. This approach suggests prioritizing the recomputation of shallower layers—an approach we refer to as Chronos-Recomp—when recomputation is considered.

Chronos-Recomp can be seamlessly built based on Chronos-Pipe. We can reserve time for recomputation just before the backward computation block of the shallow layers, as shown in Fig. 6(a). This approach introduces regular intervals during the forward pass without changing the overall pipeline scheduling. Therefore, Chronos-Recomp can also support partial recomputation by controlling the length of the interval. As discussed in the theoretical analysis of Chronos-Pipe, the intervals between Chronos-Pipe chunks depend on the value of P. As P increases, dependency conflicts may arise in this approach. However, these conflicts can be avoided by delaying the forward computation launch of chunk 2, as illustrated in Fig. 6(b) for Chronos-Recomp scheduling with P=8. This adjustment does not affect the bubble ratio and memory usage, which will be explained in the following theoretical analysis of Chronos-Recomp.

Theoretical Analysis. We analyze activation storage requirements using full recomputation of shallow layers as an example. With a chunk size of 2, the remaining activation storage is capped at only 25% of m_a. Observing micro-batch 2 in Fig. 6(b), the total intervals introduced during the forward execution of chunk 2 is $\lceil \frac{(p-1)}{2} \rceil T_{unit}$. Therefore, for Stage 0, the lifespan of chunk 2's activations is given by:

$$T_{life\_chunk2} = (3p + \lceil \frac{(p-1)}{2} \rceil - 2)T_{unit} \quad (3)$$

Here, $(3p-2)T_{unit}$ represents the ideal time from the start of chunk 2's forward computation to the arrival of the backward computation, assuming no interval is present. Moreover, since tasks in pipeline parallelism repeat with a period of $7T_{unit}$, the required number of activation storage blocks for chunk 2 is $\lceil \frac{T_{life\_chunk2}}{7T_{unit}} \rceil$, which simplifies to $\lfloor \frac{p}{2} \rfloor$. Thus, with Chronos-Recomp's full recomputation for chunk 1, the remaining activation storage is only 25%

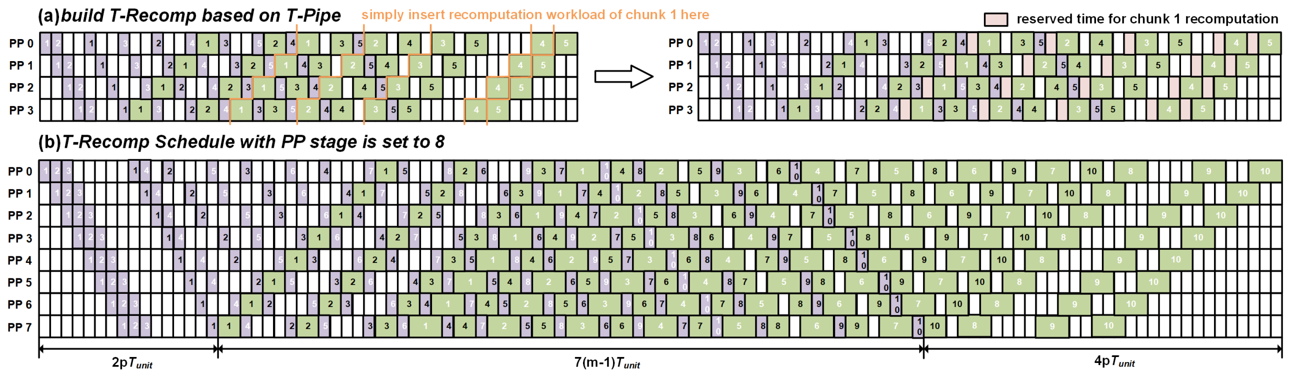

(a) build T-Recomp based on T-Pipe
simply insert recomputation workload of chunk 1 here
reserved time for chunk 1 recomputation
PP 0
PP 1
PP 2
PP 3
(b) T-Recomp Schedule with PP stage is set to 8
PP 4
PP 5
PP 6
PP 7
$2pT_{unit}$
$7(m-1)T_{unit}$
$4pT_{unit}$

Figure 6: (a) build Chronos-Recomp based on Chronos-Pipe. (b) Chronos-Recomp Schedule with PP stage is set to 8.

of m_a. By comparison, standard recomputation in 1F1B with the same computation budget requires 50% of m_a for storage, making Chronos-Recomp 1.5 times more storage-efficient than standard recomputation.

Under these conditions, we can also evaluate the total execution time, where Chronos-Recomp is similar to 1F1B with recomputation, as shown in Fig. 6(b). The warm-up phase, T_{warmup}, represents the time required to complete the forward computation of chunk 1, taking $2pT_{unit}$. The steady phase, T_{steady}, involves the forward computation of chunk 2 for the remaining $(m-1)$ microbatches, consuming $7(m-1)T_{unit}$. During the cool-down phase, the backward computation of chunk 1 for the final micro-batch requires $4pT_{unit}$. This results in a total execution time of $[6p+7(m-1)]T_{unit}$. By comparison, the execution time for 1F1B with 50% recomputation is $7(m-1+p)T_{unit}$. Notably, Chronos-Recomp's forward computation allows for overlap with half of the P2P communications. In practical scenarios, where $T_{bwd} \leq 2T_{fwd}$, the P2P communication of gradient in the backward pass of chunk 2 can overlap with recomputation. Consequently, the impact of P2P communication on Chronos-Recomp and 1F1B is similar.

As P increases, dependency conflicts can arise within the Chronos-Recomp scheduling scheme. We've analyzed this situation in detail. When P increases (e.g., $P \geq 8$), delaying the launch of the forward computation for chunk 2 effectively prevents dependency conflicts between the forward computations of the two chunks. Additionally, we demonstrate in Appendix A that for $P \leq 40$, delaying the computation of chunk 2 by just one round is sufficient to avoid forward dependency conflicts between chunks—a parallelism level that significantly exceeds typical configurations in mainstream settings. Notably, chunk 2's forward execution begins later than chunk 1's, while its backward pass completes earlier. This scheduling adjustment, therefore, does not impact the critical path and does not affect the overall execution time. Moreover, the peak activation storage analysis we conducted does not impose any limitations on the value of P, making the conclusions widely applicable across various P values.

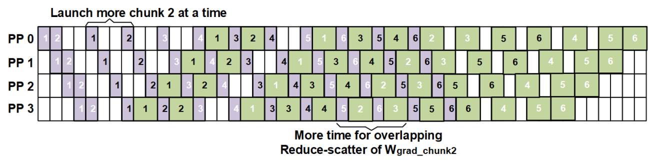

Launch more chunk 2 at a time
PP 0
PP 1
PP 2
PP 3
More time for overlapping
Reduce-scatter of W_{grad_chunk2}

Figure 7: build Zero-2-compatible schedule based on Chronos-Pipe

4.3 Compatible with ZeRo-2 Based Pipeline Schedule

Chronos-Pipe can be further adapted to support a ZeRO-2-compatible PP. As illustrated in Fig. 7, the modified Chronos-Pipe repeatedly launches the workload for the current chunk's forward computation. This increases the overlap time available for ZeRO-2's communication between DP ranks. Additionally, this adjustment has minimal impact on activation storage, enabling the launch of additional workloads of the same chunk to reduce communication demands between DP ranks. Furthermore, When recomputation needs to be considered, such as long sequence training, Chronos-Recomp can offer higher recomputation efficiency for this ZeRO-2-compatible PP scheduling approach.

5 Chronos-aware SCHEDULE FOR MODEL STATE MEMORY SAVINGS

Temporal locality can be applied not only to activation memory usage optimization but also to model state store optimization.

5.1 Chronos-Offload: Chronos-aware Offload

Chronos-Offload Scheduling Principle. The central idea behind Chronos-Offload is to offload model states with poor temporal locality to the CPU. In DNN, weights of deeper layers complete gradient computation first, allowing them to be updated earlier. However, these weights are the last to be used in the forward pass, giving them the poorest temporal locality. In the two-chunk Chronos-Pipe design, this is reflected by chunk 2 starting its forward computation later than chunk 1 but finishing its backward pass earlier. As a result, chunk 2's weight updates can be offloaded to the CPU, while chunk 1's weight updates remain on the GPU. This approach allows us to overlap chunk 2's weight update process with the idle time between the end of its backward pass in the current mini-batch and the start of its forward pass in the next mini-batch.

This approach aligns well with the Optimizer Step phase's specific computation and storage characteristics. In this phase, gradients are used to update first- and second-order momenta and the master weight (32-bit). The updated master weight is then quantized to produce a 16-bit quantized weight. Since this process involves only element-wise operations, it doesn't leverage the high computational power of Tensor Cores, resulting in inefficient GPU utilization. On the other hand, server CPUs, with their SIMD processing capabilities, are well-suited to the computational demands of the Optimizer Step. By offloading weight updates to the CPU, the first- and second-order momenta and the master weight can be kept in CPU-side DRAM, with only the quantized weight and gradients stored on the GPU. This reduces the GPU's storage requirements to roughly one-third of the total Model State. Through Chronos-Offload, we introduce temporal locality into the Model State Offload strategy and make it more suitable with PP, enabling complementary use of the CPU and GPU's respective computing and memory strengths while minimizing impact on GPU computational efficiency.

Implement of Chronos-Offload. Chronos-Pipe offers seamless compatibility with Chronos-Offload in pipeline parallelism compared to other methods. In existing approaches, the cooldown phase immediately transitions from chunk 2's backward pass to chunk 1's backward pass, eliminating scheduling bubbles. A similar issue arises in the warm-up phase, as seen in the interleaved 1F1B structure (Fig. 2(a)). Supporting Chronos-Offload in this setup requires modifying the PyTorch backend to asynchronously manage chunk 2's weight updates on the CPU, including gradient offloading, optimizer updates, and updated weight uploads. These intrusive changes increase programming complexity and risk resource conflicts, reducing compatibility with other scheduling strategies. In contrast, Chronos-Pipe naturally supports Chronos-Offload by introducing bubbles between tasks. During the cooldown phase, bubbles exist between the end of chunk 2's backward pass and the start of chunk 1's, with a similar effect in the warm-up phase. These bubbles provide ideal opportunities for Chronos-Offload without interference. This behavior stems from Chronos-Pipe's optimization of temporal activation locality, minimizing intervals between pipeline tasks while naturally creating sufficient bubbles during warm-up and cooldown phases.

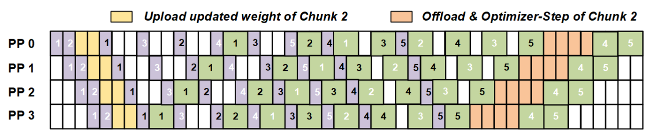

Upload updated weight of Chunk 2
Offload & Optimizer-Step of Chunk 2
PP 0
PP 1
PP 2
PP 3

Figure 8: seamless compatibility with Chronos-Offload in Chronos-Pipe

Chronos-Pipe utilizes these bubbles to enable smooth integration with Chronos-Offload, as shown in Fig. 8. During the cooldown phase, gradients of chunk 2 are offloaded, and the corresponding optimizer updates are performed in the bubbles between chunk 2's backward pass and chunk 1's. Similarly, the upload of quantized weights is scheduled during bubbles between forward passes of the two chunks in the next minibatch's warm-up phase. While this approach doesn't fully optimize Chronos-Offload's efficiency, we demonstrate that its effectiveness improves with higher levels of pipeline parallelism (PP), data parallelism (DP), and longer sequence lengths. Furthermore, the optimizer step involves synchronized operations, such as gradient norm calculations, which aggregate all gradients. Since Chronos-Offload splits optimizer updates between the CPU and GPU, synchronization overhead must be addressed. To mitigate this, a post-optimizer update validation strategy, similar to [8], can be employed to manage these challenges effectively.

Scalability with Increasing DP, PP, and Sequence Length. Two conditions must be met to complete the Chronos-Offload process within the available 'bubbles.' These conditions are more easily satisfied as the degree of PP and sequence length increase. The first condition requires the gradient offload and the CPU optimizer updates to be completed within the cooldown phase's 'bubbles.' In Chronos-Pipe, the interval between backward computations of chunks is $(3 + 6\lceil\frac{(2p-3)}{6}\rceil - 2p)T_{unit}$, which results in a delay of $\lceil\frac{(2p-3)}{6}\rceil$ rounds for chunk 1 relative to chunk 2. This aligns the backward computation of the first micro-batch for chunk 1 with the $(1+\lceil\frac{(2p-3)}{6}\rceil)$ th one of chunk 2. Thus, we define:

$$T_{avaliable\_offload} = (p - \lceil\frac{(2p-3)}{6}\rceil - 1)\frac{T_{bwd}}{2p} \tag{4}$$

Here, $T_{avaliable\_offload}$ represents the idle time available between the end of the backward computation for chunk 2 and the start of backward computation for chunk 1 during the cooldown phase. Ideally, this time would be sufficient to complete the gradient offload and the CPU update calculations. It follows that:

$$\frac{T_{step}}{2p} \leq T_{avaliable\_offload} \Rightarrow \frac{T_{step}}{T_{bwd}}\frac{1}{(p - \lceil\frac{(2p-3)}{6}\rceil - 1)} \leq 1 \tag{5}$$

Where T_{step} denotes the total time required for offloading gradients in all layers of DNN and performing CPU optimizer updates. As P increases, the impact of the constant terms decreases, simplifying the equation to $\frac{T_{step}}{T_{bwd}}\frac{3}{2p} \leq 1$ for large P, nearly offering a linear increase in idle time, thus facilitating easier fulfillment of this condition. The ratio $\frac{T_{step}}{T_{bwd}}$ is constant for a given model and is influenced by factors such as sequence length, batch size, offload communication bandwidth, and CPU processing speed. Longer sequences and larger batch sizes further decrease $\frac{T_{step}}{T_{bwd}}$, linearly increasing the idle time available for Chronos-Offload, even without considering the attention operator.

The second requirement is that quantized weight uploads must be completed within the 'bubbles' of the warm-up phase. Utilizing a similar analytical approach, we observe that the forward pass of chunk 2 is delayed by $\lceil\frac{(p-3)}{6}\rceil$ rounds relative to chunk 1. Consequently, we derive:

$$T_{avaliable\_upload} = (p - \lceil \frac{(p-3)}{6} \rceil - 1) \frac{T_{fwd}}{2p} \quad (6)$$

$$\frac{T_{upload}}{2p} \leq T_{avaliable\_upload} \Rightarrow \frac{T_{upload}}{T_{fwd}} \frac{1}{(p - \lceil \frac{(p-3)}{6} \rceil - 1)} \leq 1 \quad (7)$$

Here, $T_{avaliable\_upload}$ represents the idle time during the warm-up phase available for uploading quantized weights, and T_{upload} denotes the total time required to upload quantized weights of all layers. As P increases significantly, this relationship simplifies to $\frac{T_{upload}}{T_{fwd}} \frac{6}{5p} \leq 1$. Similar to the first condition, this requirement becomes easier to meet with increased pipeline stages p and longer sequence lengths. Thus, Chronos-Offload exhibits strong scalability with increasing degrees of pipeline parallelism (PP) and sequence lengths, which is suitable for future development trends.

Additionally, Chronos-Offload scales effectively with increased Data Parallelism (DP). As the level of DP increases, the amount of weights each DP rank needs to update decreases, which in turn reduces the time required for gradient offloading and optimizer computations.

5.2 Compatible with Other Schedule

Chronos-Offload is designed to be compatible with a variety of scheduling strategies. Our approach exclusively leverages the 'bubbles' created within Chronos-Pipe, eliminating competition over communication bandwidth and CPU with other scheduling approaches. As a result, it seamlessly supports many parallelism strategies, including tensor parallelism, context parallelism, and data parallelism. This flexibility makes it widely applicable across diverse distributed parallelism training frameworks.

Chronos-Offload is also fully compatible with the CPU Activation Offloading scheme. It performs Model State Offloading in the interval between the completion of chunk 2's backward computation and the onset of the forward computation for the next mini-batch of chunk 2. This process exclusively utilizes the PCIe upload bandwidth during the warm-up phase and the PCIe offload bandwidth during the cool-down phase. This approach does not interfere with CPU Activation Offloading; rather, it enhances the efficient utilization of the PCIe bidirectional bandwidth.

Therefore, Chronos-Offloading effectively utilizes the temporal locality of weights while also maximizing the use of the prevalent 'bubbles' during pipeline parallelism's warm-up and cool-down phases. Consequently, it can seamlessly integrate with numerous scheduling optimization strategies.

6 EXPERIMENTAL RESULTS

In this section, we seek to answer the following questions:

• How well does ChronosPipe perform in memory and throughput?How does it compare with other recomputation-based solutions?

• How does ChronosPipe perform across varying sequence lengths and model architectures (input embedding, attention, activation function)?

• What is the actual impact of P2P communication on ChronosPipe? Can the scalability of Chronos-Offload truly maintain performance without degradation?

• What potential benefits could be achieved by further leveraging temporal locality in ChronosPipe?

6.1 Methodology

Setup. Our experiments were carried out on a cluster comprising up to 64 GPUs, organized across 8 nodes. Nodes are interconnected with four 200 Gbps NICs. Each node is equipped with 8 GPUs and 2 CPUs(Intel Xeon Platinum 8480+) and 2TB DRAM, 15.36TB NVME. Each GPU features 32GB of HBM and connects to the CPU through PCIe5 x8(32GB/s), with high-bandwidth interconnects linking GPUs within a node.

Workloads. For these experiments, we utilized a model architecture similar to LLAMA2-70B [45], consisting of 80 transformer layers and incorporating the GQA mechanism. We varied the number of transformer layers in the model to adjust the size for different experiments.

We further analyze the effectiveness of ChronosPipe on other models through theoretical analysis. Qwen2.5-32B [48], PaLM-62B [47], and OPT-66B [46] are selected as they have distinct vocabulary sizes, attention mechanism configurations, and activation functions, being typical enough for current dense models.

Baselines. We compare ChronosPipe against 1F1B and interleave-1F1B to highlight the benefits of introducing temporal locality into pipeline parallelism. All scheduling strategies were implemented within a framework similar to Megatron-LM to ensure a fair comparison. In ChronosPipe's implementation, only essential asynchronous communications were employed, while all other P2P communications followed the synchronous approach used in the baselines. When recomputation is included, we denote it using "scheduling strategy + recomputation ratio." For instance, $1F1B + R = 50\%$ represents the 1F1B with a 50% recomputation ratio.

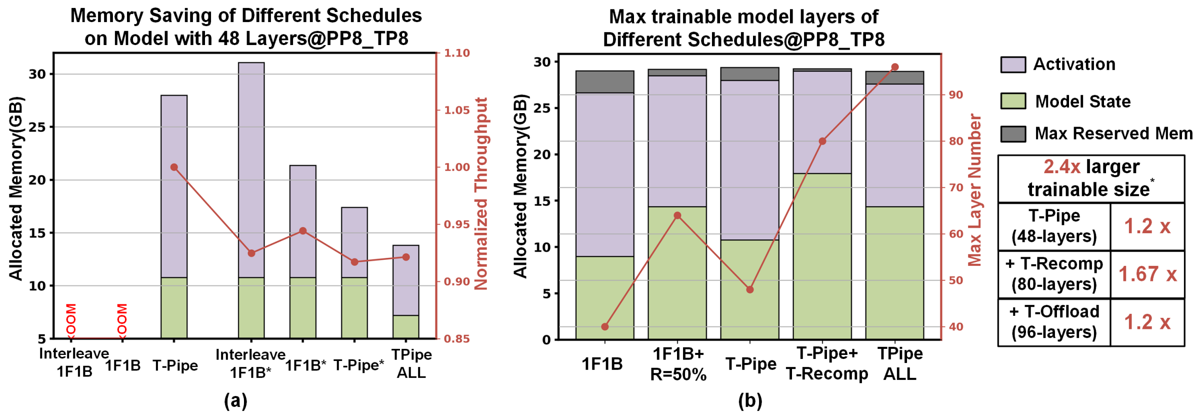
Additionally, through theoretical analysis, we further compare our approach with the recomputation strategies of Megatron-Kwai [11] and AdaPipe [27]. Conditions are aligned with AdaPipe's publicly - available data for a fair comparison.

Unless otherwise specified, the input of ColumnLinear is not evenly divided along the TP dimension, consistent with Megatron's settings. Meanwhile, Operator-level recomputation (RMSNorm, activation function) and FlashAttention [44] are enabled by default. In theoretical calculations, $T_c = 0.104 T_{unit}$ is set.

Theoretical analysis. Based on the following two reasons, we assert that theoretical analysis demonstrates sufficient credibility in computational and storage evaluation. First, the performance of PP can be effectively modeled. By precisely quantifying the bubble ratio and recomputation proportion, accurate performance estimation becomes achievable—an approach validated as feasible in [9, 43]. Regarding storage, Activation and Model State dominate storage consumption, enabling theoretical analysis to effectively identify critical storage constraints.

6.2 End-to-End Evaluation

Fig. 9(a) illustrates that ChronosPipe significantly reduces HBM memory requirements while maintaining comparable throughput.



2.4x larger trainable size*	
T-Pipe (48-layers)	1.2 x
+ T-Recomp (80-layers)	1.67 x
+ T-Offload (96-layers)	1.2 x

Figure 9: End-to-End Evaluation on ChronosPipe. (a) shows the memory saving of different schedules on a 48-layer model at (PP, TP)=(8,8), global batch=128, micro batch=2, seq len=4K. *: R=66.7% in interleave 1F1B, R=50% in 1F1B and Chronos-Recomp is used in Chronos-Pipe. (b) illustrates the maximum trainable model size of different schedules at (PP, TP)=(8,8), global batch=128, micro batch=2, seq len=4K. *: compared with 1F1B.

Under the PP8_TP8 configuration, ChronosPipe's optimized scheduling enables training a 48-layer (i.e., 42B) model, while Interleave-1F1B and 1F1B run into out-of-memory (OOM) issues. Chronos-Recomp further demonstrates improved recomputation efficiency, reducing activation memory from 17.23 GB to 6.63 GB. In contrast, under the same recomputation budget (R=50%), the 1F1B method still occupies 10.6 GB for activations. With Chronos-Offload, ChronosPipe ALL reduces the total memory footprint for activations and model states to 64.6% of what is used in the $1F1B{+}R = 50\%$ setup while achieving 97.58% of its throughput.

Fig. 9(b) explicitly evaluates the effect of ChronosPipe in terms of the maximum size of the trainable model. With limited 32GB HBM, 1F1B can train only a 40-layer model under the PP8_TP8 configuration, while training a 64-layer model requires a 50% recomputation ratio. In contrast, ChronosPipe can train a 48-layer model, and with the Chronos-Recomp strategy, this can be extended to 80 layers. Further leveraging Chronos-Offload enables training even a 96-layer model, all while keeping the max reserved memory at roughly the same level. As a result, ChronosPipe increases the scale of trainable models to 2.4 times the 1F1B, and 1.5 times that of the 1F1B+R=50% configuration. Note that this only shows the advantage of introducing temporal locality into PP. ChronosPipe can further integrate other compatible optimization strategies to optimize HBM usage, which is not reflected in this experiment.

Fig. 10 compares ChronosPipe with other recomputation solutions. Megatron-Kwai's Operator-aware recomputation reduces storage by 1.27x with negligible performance loss. Applying Operator-aware recomputation to remaining activations shows diminishing returns, but co-designing PP with recomputation can enhance its benefits. AdaPipe adopts Stage-aware task and recomputation budget allocation across pipeline stages, achieving 1.76x storage savings and 1.26x performance improvement over the 1F1B+R = 100% baseline. Without Operator-aware recomputation on activations of deeper layers, Chronos-Pipe+Chronos-Recomp attains 1.72x storage savings and 1.17x performance gain relative to the same baseline. ChronosPipe ALL further achieves 2.22x storage savings, highlighting the efficiency of Locality-aware design. While Chronos-Pipe+Chronos-Recomp exhibits moderately reduced throughput relative to AdaPipe, it provides a general and simple solution, and enhance adaptability to a variety of changes including model scale, structures, and parallelism configurations. In contrast, AdaPipe necessitates sophisticated search mechanisms and programmatic overhead to implement stage-aware task and recomputation budget

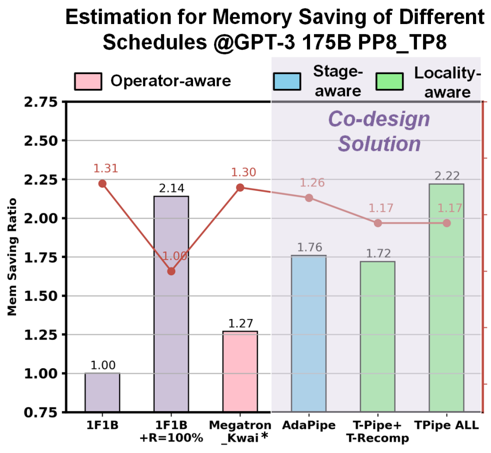


Figure 10: Estimated memory saving for different recomputation-based solutions on GPT3-175B at (PP,TP)=(8,8), global batch=32, micro batch=1, seq len=16K. sequence parallelism (SP) [17] is used in all and operator-level recomputation(RMSNorm, activation function) is not used in ChronosPipe. *:1F1B with only Operator-level recomputation solution is used. activation-offloading is not under consideration due to it suitable for other solutions.

allocation, which is a hard integer partitioning problem. Maintaining workload equilibrium across varying constraints mentioned while preserving throughput remains questionable. Moreover, these approaches embody distinct design philosophies, revealing synergistic potential for future integration.

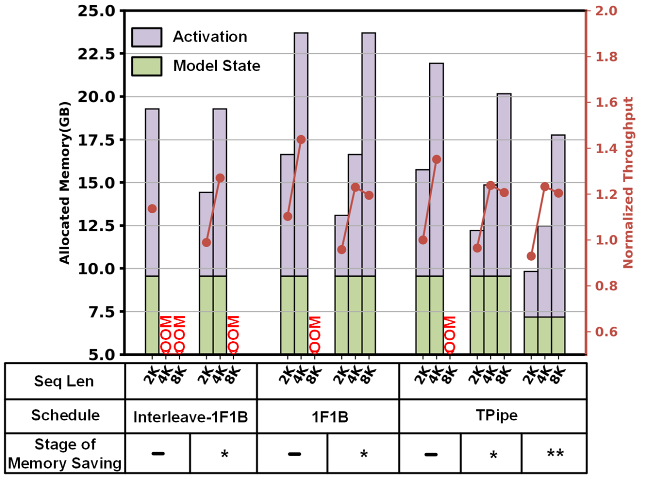


Figure 11: memory and throughput of different schedules under various sequence lengths on a 32-layer model at (DP, PP, TP)=(2,4,8), global batch=128, micro batch=2. *: R=50%, Chronos-Recomp is used in ChronosPipe. **: Chronos-Offload is further adopted.

We further study the behavior of different scheduling strategies across various sequence lengths, shown in Fig. 11. Under the DP2 (ZeRO-1)_PP4_TP8 configuration, Interleave-1F1B exhibits the worst memory efficiency, running out of memory (OOM) at a sequence length of 8k, even with R=50%. Theoretical analysis suggests that smaller PP reduces ChronosPipe's memory efficiency, resulting in Chronos-Pipe and Chronos-Recomp saving only 12.5% and 25% of activation respectively compared to 1F1B and its recomputation variants in this setup. However, as sequence lengths increase, ChronosPipe's memory savings become increasingly pronounced, demonstrating superior memory efficiency for longer sequences. Additionally, Chronos-Offload further enhances ChronosPipe's memory efficiency by reducing model state memory, allowing ChronosPipe to excel as both sequence length and model size scale up. In terms of throughput, while Chronos-Pipe experiences a relative throughput drop of 6%–9% compared to 1F1B, both Chronos-Recomp, and ChronosPipe ALL maintain throughput comparable to 1F1B+R=50% across all sequence lengths, even when throughput across sequence lengths are slightly influenced by Operator Library.

As shown in Fig. 12, we further evaluated the performance of ChronosPipe on dense models under multiple different parameter configurations. Chronos-Pipe+Chronos-Recomp achieves a storage reduction of 1.21x-1.26x while maintaining a throughput similar to that of 1F1B+R=50%. This enables the training of PaLM-62B and OPT-66B models within the 32GB HBM constraint. Moreover,

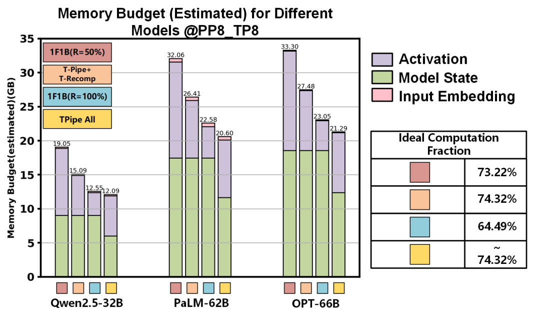


Figure 12: estimated memory for different models at (PP,TP)=(8,8), global batch=128, micro batch=2. Ideal Computation Fraction is defined with following equation:(1- bubble overhead - recomputation overhead).

ChronosPipe ALL achieves a storage reduction of 1.56x-1.58x compared to 1F1B+R=50%. Compared to 1F1B+R=100%, ChronosPipe not only boosts the throughput by approximately 1.15x but also reduces the storage by 1.04x-1.10x.

6.3 Further Analysis of ChronosPipe

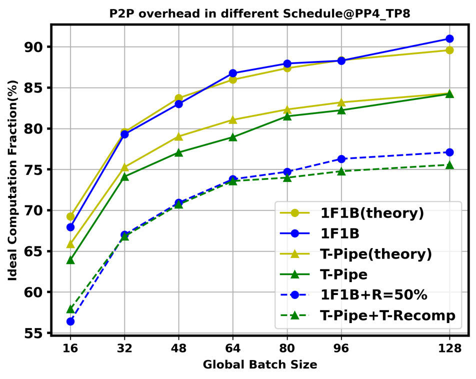
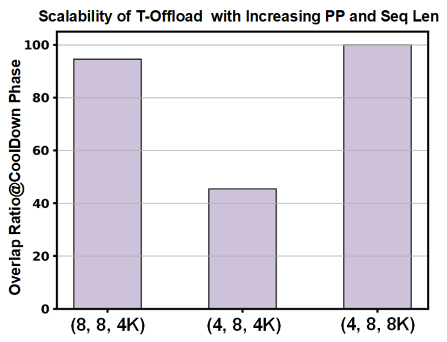


Figure 13: analysis on P2P overhead on 24-layer model at (PP,TP)=(4,8), micro batch=2, seq len=4K. Ideal Computation Fraction is defined with following equation:(1- bubble overhead - recomputation overhead).

As illustrated in Fig. 13, we evaluated the impact of P2P communication on ChronosPipe under a PP4_TP8 configuration with a 24-layer model. During execution, Chronos-Pipe showed approximately a 6% lower ideal computation fraction compared to 1F1B. The theoretical analysis attributes 5% of this reduction to the overhead introduced by P2P communication, indicating that the primary gap stems from the additional round of P2P communication. This significant P2P overhead can be attributed to low P2P bandwidth and frequent P2P communication(one P2P communication happens after every three layers of computation in this setup), leading to $T_c \approx 0.104 T_{unit}$. Despite this, Chronos-Recomp achieves an ideal computation fraction comparable to 1F1B+R=50% ($\leq$ 3%), suggesting that its overall P2P cost is similar to that of 1F1B. Specifically, Chronos-Recomp overlaps roughly half of the forward pass P2P communication with computation. However, due to the limited number of layers in the workload, gradient P2P communication during the backward pass (in chunk 2) does not overlap effectively with recomputation, resulting in minor degradation. In practice, ChronosPipe's superior memory efficiency allows it to train larger workloads, naturally reducing the relative impact of P2P communication. As the workload scales, Chronos-Recomp achieves better overlap for gradient P2P communication during the backward pass, further mitigating its impact on performance.



(PP,TP,Seq Len)	(8, 8, 4K)	(4, 8, 4K)	(4, 8, 8K)
Norm.Time w/o T-Offload	1.000	1.276	2.505
Norm.Time w T-Offload	0.994	1.280	2.497

Figure 14: evaluate the scalability of Chronos-Offload on a 16-layer model at global batch=128, micro batch=2.

We further verify the scalability of Chronos-Offload, as illustrated in Fig. 14. When training a 16-layer model with a sequence length of 4K on PP4_TP8 setup, Chronos-Pipe achieves only partial overlap (45.45%) between the bubble created during the cooldown phase and the time required for weight gradient offloading and CPU-based optimizer updates. This limited overlap results in a slight increase in overall training time. However, when either the pipeline stage or the sequence length is doubled, the overlap improves significantly to 94.55% and 100%, respectively. Furthermore, Chronos-Offload enables only half of the optimizers to be updated on GPU, leading to a slight decrease in overall training time. These results highlight the scalability of Chronos-Offload, demonstrating its increasing effectiveness as the PP stage and sequence length grows, making it a viable solution for handling larger workloads.

6.4 Further Design Space Exploration

Fig. 15 illustrates our in-depth analysis of Chronos-Recomp. We theoretically evaluated Chronos-Recomp performance under PP4_TP8 configuration with chunk sizes of 2, 3, and 4. While small PP is less favorable for larger chunks in Chronos-Pipe, a clear trend is observed: Chronos-Recomp prioritizes recomputation tasks based on their temporal locality. When the recomputation budget is limited, focusing recomputation on shallower layers provides higher gain. For example, with a chunk size of 4, recomputing only 25% of the layers can reduce activation memory usage by up to 43.75%.

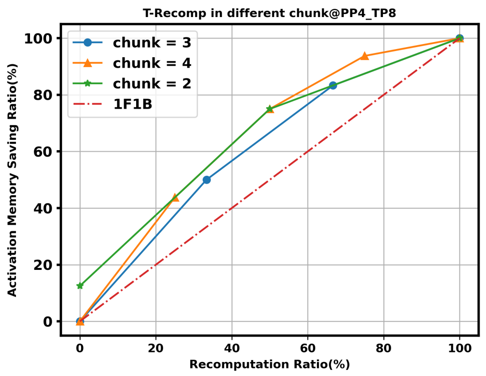


Figure 15: in-depth analysis of Chronos-Recomp. Additional memory required for recomputation is ignored for better illustration.

However, as the recomputation ratio increases, Chronos-Recomp incurs a higher cost for recomputation in layers where it is less efficient. Nevertheless, Chronos-Recomp consistently outperforms standard recomputation strategies, as shown in Fig. 15.

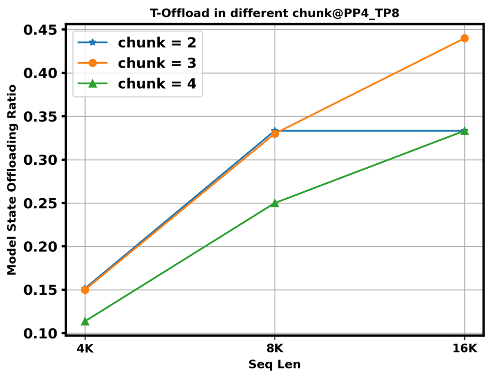


Figure 16: in-depth analysis of Chronos-Offload.

When partitioning the workload into more chunks, Chronos-Offload has the potential to offload more model states. Our theoretical analysis, illustrated in Fig. 16, explores this behavior while maintaining throughput. The results indicate that Chronos-Offload experiences diminishing returns as the number of chunks increases; in some cases, memory efficiency can even decrease. This effect arises because adding more chunks only marginally increases the number of usable bubbles in the cooldown phase, particularly under smaller PP configurations. For example, when the PP stage is set to 4, using chunk=3 only provides 50% more bubbles compared to chunk =2, while chunk =4 produces the same number of bubbles as chunk =3. Consequently, the advantages of chunk =3 are mainly evident with long sequence length, whereas chunk =4 underperforms because the number of chunks matches the PP stage, ultimately reducing overall memory efficiency.

7 RELATED WORK

Hybrid Parallelism. To address the challenges of scaling model size and sequence length, modern distributed training systems leverage GPU clusters to expand overall memory capacity while deploying various schedules. These strategies include data parallelism(DP), pipeline parallelism(PP), tensor parallelism(TP), context parallelism(CP), and so on. DP [13, 19], PP [4–8], and TP [20–23] enable the distribution of model states across machines. CP [24, 25] and sequence parallelism(SP) [17] specialize in distributing activations across their respective dimensions. Hybrid parallelism combines these approaches, enabling efficient scaling to models with trillions of parameters. PP is an essential component of hybrid parallelism among these strategies, particularly in cross-node deployments, due to its low communication requirement. However, PP suffers from pipeline bubbles and high activation memory overhead. Moreover, it is often less compatible with advanced DP like ZeRO-2/3.

Pipeline Parallelism. Most pipeline parallelism research focuses on reducing pipeline bubbles by finer-grained task partitioning [4–8] or multi-dimensional task scheduling [9, 10]. However, they neglect the significant overhead of activation memory. [26] reduces activation lifespans through scheduling but risks introducing more bubbles and prolonging gradient lifespan. It's V-shaped scheduling also complicates compatibility with ZeRO-2/3 and Chronos-Offload, making storage optimization for other data more challenging. [27] addresses activation memory imbalance with asymmetric partitioning but increases programming complexity. [14, 15], achieve ZeRO-2/3-compatibile PP at the cost of higher activation memory. [3] takes advantage of [14] and 1F1B for improved ZeRO-2 compatibility but only maintains, rather than reduces, activation memory usage.

Recomputation and Offloading. In addition to optimizing multi-GPU systems' HBM utilization through hybrid parallelism, memory requirements can also be reduced by trading memory for computation or communication time. For recomputation, some studies [11, 17, 50, 51] focus on Selective recomputation of different operators but fail to address the high memory usage of the projection operator. [49] introduced temporal locality-aware recomputation principle and demonstrated its efficacy in DP, but coarse-grained PP and risk of inducing steady-phase bubbles limit the application of this principle in PP. Some approaches leverage the characteristics of Hybrid parallelism. [27] adapts recomputation to mitigate activation memory imbalance in PP. [28] overlaps recomputation with collective communication in the TP dimension to reduce overhead. However, these approaches increase programming complexity. For offloading, [12] addresses memory imbalance in PP by offloading Activation across HBM. Most studies focus on fully utilizing storage systems. For example, [18, 29, 30] explore offloading activation and model states to DRAM, while [31–34] extend this to NVMe. Due to limited bandwidth in offloading, these efforts primarily focus on identifying opportunities to overlap offloading with computation and realize near-data processing.

8 CONCLUSION

This work introduces ChronosPipe to address the memory capacity limitations in LLM training. The core idea behind ChronosPipe is to treat HBM as a fast but limited "cache," optimizing and leveraging temporal locality in LLM pretraining for efficient HBM usage. Chronos-Pipe enhances activation temporal locality in PP scheduling and, together with Chronos-Recomp and Chronos-Offload, discards data with poor temporal locality from HBM. Experiments show that ChronosPipe can expand the trainable model size by 2.4x while maintaining comparable throughput, achieving 1.5x better than the 1F1B strategy combined with recomputation. At the same time, ChronosPipe has good enough compatibility to make it suitable for large-scale hybrid parallelism pre-training.

References

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[2] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[3] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[4] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," 2019. [Online]. Available: https://arxiv.org/abs/1811.06965

[5] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia, L. Diao, X. Liu, and W. Lin, "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*, ser. PPoPP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 431–445. [Online]. Available: https://doi.org/10.1145/3437801.3441593

[6] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "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*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476209

[7] Z. Liu, S. Cheng, H. Zhou, and Y. You, "Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–13.

[8] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero bubble pipeline parallelism," *arXiv preprint arXiv:2401.10241*, 2023.

[9] S. Li and T. 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*, 2021, pp. 1–14.

[10] W. Zhang, B. Zhou, X. Tang, Z. Wang, and S. Hu, "Mixpipe: Efficient bidirectional pipeline parallelism for training large-scale models," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[11] T. Yuan, Y. Liu, X. Ye, S. Zhang, J. Tan, B. Chen, C. Song, and D. Zhang, "Accelerating the training of large language models using efficient activation rematerialization and optimal hybrid parallelism," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 545–561.

[12] T. Kim, H. Kim, G.-I. Yu, and B.-G. Chun, "Bpipe: memory-balanced pipeline parallelism for training large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 16 639–16 653.

[13] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[14] J. Lamy-Poirier, "Breadth-first pipeline parallelism," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 48–67, 2023.

[15] D. Tang, L. Jiang, J. Zhou, M. Jin, H. Li, X. Zhang, Z. Pei, and J. Zhai, "Zeropp: Unleashing exceptional parallelism efficiency through tensor-parallelism-free methodology," 2024. [Online]. Available: https://arxiv.org/abs/2402.03791

[16] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[17] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 341–353, 2023.

[18] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "Zero-offload: Democratizing billion-scale model training," 2021. [Online]. Available: https://arxiv.org/abs/2101.06840

[19] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "Pytorch fsdp: Experiences on scaling fully sharded data parallel," 2023. [Online]. Available: https://arxiv.org/abs/2304.11277

[20] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Accpar: Tensor partitioning for heterogeneous deep learning accelerators," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 342–355.

[21] Q. Xu, S. Li, C. Gong, and Y. You, "An efficient 2d method for training super-large deep learning models," 2021. [Online]. Available: https://arxiv.org/abs/2104.05343

[22] B. Wang, Q. Xu, Z. Bian, and Y. You, "Tesseract: Parallelize the tensor parallelism efficiently," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[23] Z. Bian, Q. Xu, B. Wang, and Y. You, "Maximizing parallelism in distributed training for huge neural networks," *arXiv preprint arXiv:2105.14450*, 2021.

[24] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," *arXiv preprint arXiv:2310.01889*, 2023.

[25] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, S. L. Song, S. Rajbhandari, and Y. He, "Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models," *arXiv preprint arXiv:2309.14509*, 2023.

[26] P. Qi, X. Wan, N. Amar, and M. Lin, "Pipeline parallelism with controllable memory," *arXiv preprint arXiv:2405.15362*, 2024.

[27] Z. Sun, H. Cao, Y. Wang, G. Feng, S. Chen, H. Wang, and W. Chen, "Adapipe: Optimizing pipeline parallelism with adaptive recomputation and partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 86–100.

[28] P. Chen, W. Zhang, S. He, Y. Gu, Z. Peng, K. Huang, X. Zhan, W. Chen, Y. Zheng, Z. Wang, Y. Yin, and G. Chen, "Optimizing large model training through overlapped activation recomputation," 2024. [Online]. Available: https://arxiv.org/abs/2406.08756

[29] J. Fang, Z. Zhu, S. Li, H. Su, Y. Yu, J. Zhou, and Y. You, "Parallel training of pre-trained models via chunk-based dynamic memory management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 304–315, 2022.

[30] Y. Feng, M. Xie, Z. Tian, S. Wang, Y. Lu, and J. Shu, "Mobius: Fine tuning large-scale models on commodity gpu servers," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 489–501.

[31] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2021, pp. 1–14.

[32] X. Nie, Y. Liu, F. Fu, J. Xue, D. Jiao, X. Miao, Y. Tao, and B. Cui, "Angel-ptm: A scalable and economical large-scale pre-training system in tencent," *arXiv preprint arXiv:2303.02868*, 2023.

[33] H. Jang, J. Song, J. Jung, J. Park, Y. Kim, and J. Lee, "Smart-infinity: Fast large language model training using near-storage processing on a real system," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 345–360.

[34] C. Liao, M. Sun, Z. Yang, K. Chen, B. Yuan, F. Wu, and Z. Wang, "Adding nvme ssds to enable and accelerate 100b model fine-tuning on a single gpu," *arXiv preprint arXiv:2403.06504*, 2024.

[35] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[36] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[37] S. W. Yoon, J. H. Ku, N. Suthiwongsunthorn, P. C. Marimuthu, and F. Carson, "Fabrication and packaging of microbump interconnections for 3d tsv," in *2009 IEEE International Conference on 3D System Integration*. IEEE, 2009, pp. 1–5.

[38] N. Khan, V. S. Rao, S. Lim, H. S. We, V. Lee, X. Zhang, E. B. Liao, R. Nagarajan, T. C. Chai, V. Kripesh, and J. H. Lau, "Development of 3-d silicon module with tsv for system in packaging," *IEEE Transactions on Components and Packaging Technologies*, vol. 33, no. 1, pp. 3–9, 2010.

[39] H. Lee, J. Kim, M.-K. Kim, W. Lee, A. Jang, H. Lee, and D.-W. Kim, "A study on d2w hybrid cu bonding technology for hbm multi-die stacking," in *2024 IEEE 74th Electronic Components and Technology Conference (ECTC)*. IEEE, 2024, pp. 76–80.

[40] K. Kim, S. Lim, D. Jung, J. Choi, S. Na, J. Yeom, M. Lee, J. Kim, J. Kwon, K.-I. Moon, G. Lee, and K. Lee, "C2w hybrid bonding interconnect technology for higher density and better thermal dissipation of high bandwidth memory," in *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*. IEEE, 2023, pp. 1048–1052.

[41] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[42] "Nvidia h100 tensor core gpu architecture," 3 2022, [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

[43] K. Osawa, S. Li, and T. Hoefler, "Pipefisher: Efficient training of large language models using pipelining and fisher information matrices," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 708–727, 2023.

[44] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in neural information processing systems*, vol. 35, pp. 16 344–16 359, 2022.

[45] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[46] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[47] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[48] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei *et al.*, "Qwen2. 5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[49] M. Kirisame, S. Lyubomirsky, A. Haan, J. Brennan, M. He, J. Roesch, T. Chen, and Z. Tatlock, "Dynamic tensor rematerialization," *arXiv preprint arXiv:2006.09616*, 2020.

[50] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: Dynamic gpu memory management for training deep neural networks," in *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, 2018, pp. 41–53.

[51] X. Peng, X. Shi, H. Dai, H. Jin, W. Ma, Q. Xiong, F. Yang, and X. Qian, "Capuchin: Tensor-based gpu memory management for deep learning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 891–905.

A Formalization on Delay Rounds at Chronos-Recomp

In Chronos-Pipe scheduling, an additional interval time $T_{fwd\_interval}$ exists between forward chunks, derived from steps 1, 2, and 3 in Fig. 5(a), resulting in $(3+6\lceil\frac{(p-3)}{6}\rceil-p)T_{unit}$. In Chronos-Recomp's shallow-layer full recomputation, the execution times for steps 1, 2, and 3 respectively change T_{unit}, $\lceil\frac{(p-3)}{6}\rceil T_{unit}$, and $\lceil\frac{(p-1)}{2}\rceil T_{unit}$. Therefore, $\Delta T_{fwd\_interval}$ becomes $(\lceil\frac{(p-1)}{2}\rceil-\lceil\frac{(p-3)}{6}\rceil-1)T_{unit}(P\geq 3)$. Based on this, We can formalize the number of rounds k required to delay the launch of chunk 2 as an optimization problem for $P \geq 3$:

$$\begin{aligned} &Min \quad k \\ s.t. \quad &T_{fwd\_interval}-\Delta T_{fwd\_interval}+7kT_{unit}\geq 0 \\ &k\in N \\ &p\geq 3 \end{aligned} \tag{8}$$

Therefore, when $8 \leq p \leq 40$, we have $k = 1$, meaning that chunk 2 needs to be delayed by one round.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009