Liger: Interleaving Intra- and Inter-Operator Parallelism for Distributed Large Model Inference

Jiangsu Du
Sun Yat-sen University
dujiangsu@mail.sysu.edu.cn

Jinhui Wei
Sun Yat-sen University
weijh28@mail2.sysu.edu.cn

Jiazhi Jiang
Sun Yat-sen University
jiangjzh6@mail2.sysu.edu.cn

Shenggan Cheng
National University of Singapore
shenggan@comp.nus.edu.sg

Dan Huang
Sun Yat-sen University
huangd79@mail.sysu.edu.cn

Zhiguang Chen*
Sun Yat-sen University
chenzhg29@mail.sysu.edu.cn

Yutong Lu
Sun Yat-sen University
luyutong@mail.sysu.edu.cn

Abstract

Distributed large model inference is still in a dilemma where balancing cost and effect. The online scenarios demand intra-operator parallelism to achieve low latency and intensive communications makes it costly. Conversely, the inter-operator parallelism can achieve high throughput with much fewer communications, but it fails to enhance the effectiveness.

In this paper, we present Liger, a distributed large model inference runtime system that is of capability to achieve low latency at high throughput on the multi-GPU architecture. The key idea lies in the novel interleaved parallelism, which interleaves the computation and communication across requests. Liger enables this parallelism by carefully scheduling computation and communication kernels across requests onto multiple streams of multiple GPUs. It achieves precise control of kernel execution order efficiently by mixing use the CPU-GPU synchronization and the inter-stream synchronization. To prevent scheduling failures caused by resource contention, Liger introduces a contention factor strategy to anticipate the penalty of contention. It enables a higher degree of overlap by decomposing lengthy kernels into smaller, more manageable units at runtime.

Extensive evaluations show that Liger, in most cases, outperforms existing parallelism approaches across models and devices, presenting the best latency and throughput results. In a 4-device case, Liger reduces the average latency by 36.0% while maintaining the same throughput compared to the inter-operator approach. Meanwhile, it improves the throughput by 1.34× with improved average latency compared to the intra-operator approach.

CCS Concepts: • **Computing methodologies** → **Distributed algorithms**; **Distributed artificial intelligence**.

Keywords: Distributed Large Model Inference, GPU Scheduling, Multi-GPU Architecture

1 Introduction

Advances in self-supervised learning promotes exponential scaling in the sizes of models. Large models such as GPT [3], OPT [41], and GLM [9] have successfully unlocked a broad spectrum of intelligent applications, from AI programmer [13] to chatbot [27].

As their deployment demands surge, the high computational and memory requirements of large models pose significant challenges for the inference serving infrastructure, particularly in online scenarios, or rather latency-critical situations. On one hand, to enhance user experience, low latency is crucial, which refers to the time interval between a job's arrival to its completion. On the other hand, high throughput, which refers to the number of requests a system can handle within a given time, is essential to meet the soaring demand. Techniques like model compression are not always sufficient to address the challenges and often cause the reduced model quality [8]. Therefore, enterprises adopt distributed inference that leverages multiple advanced devices (GPUs) to provide a single instance with more memory capacity and computing resources for serving large model applications. However, existing parallelism approaches for distributed large model inference face a dilemma in balancing cost and effect.

*Corresponding authors.

PPoPP '24, March 2–6, 2024, Edinburgh, United Kingdom

ACM ISBN 979-8-4007-0435-2/24/03…$15.00
https://doi.org/10.1145/3627535.3638466

To achieve computation parallelization and distributed memory allocation among devices, existing distributed inference systems [2, 23, 39] employ two types of parallelism approaches: intra-operator parallelism [5, 35] and inter-operator parallelism [16, 18]. They have been extensively studied in training tasks which is throughput-oriented. Both approaches divide the target model into fragments and allocate each fragment to a separate device, allowing the serving system to process larger models. However, it is noted that these two approaches have distinct impacts on latency and throughput.

The intra-operator approach partitions each operation across multiple devices. It can reduce latency for the computation of a single operation is performed on multiple devices concurrently. To eliminate data dependencies between devices, it necessities frequent collective communications. Although the intra-operator approach is capable of reducing latency and providing engaging user experience, such a high communication traffic severely damages throughput, leading to a higher cost for a single request.

Alternatively, the inter-operator approach partitions the target model into disjoint stages, each stage includes a consecutive set of operations and stays on a separate device. Requests are then processed in a pipelined manner. Limited communications make throughput increase linearly with the device number and it can process a single request cheaper. However, it fails to reduce latency as each request is processed by a single device at a time. Consequently, existing distributed inference systems are stuck in the dilemma where balancing cost and effect, limited by the fundamental features of intra- and inter-operation parallelism. Detailed analysis with case studies is shown in §2.2.

To mitigate the dilemma, we propose Liger, a runtime system which is capable of achieving low latency at high throughput for distributed large model inference. Specifically, Liger focuses on the multi-GPU architecture where a single node is equipped with multiple GPUs and it is noted that this architecture can cover the vast majority of model sizes. Liger is built upon a newly proposed parallelism, the interleaved parallelism, which has advantages of both intra- and inter-operator parallelism. More specifically, it interleaves the computation and communication across multiple requests. Liger enables this novel parallelism by ingeniously scheduling inference kernels of different types in GPUs.

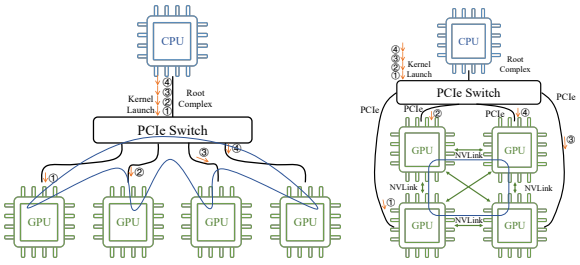
Efficiently scheduling and overlapping computation and communication kernels on multiple GPUs face challenges from various aspects: 1) how to precisely and efficiently control kernel execution order, 2) how to handle the resource contention problem, 3) how to increase the extend of overlap.

First is how to precisely and efficiently control kernel execution order of computation and communication kernels across multiple streams. The underlying scheduling logic in GPUs can lead to uncertain execution order when launching a large number of kernels, particularly with communication kernels, onto multiple streams. By contrast, relying on frequent acquisition of the GPU status to determine the launch time of kernels will introduce overhead. To address this conflict, Liger achieves precise control by employing a combination of CPU-GPU synchronization and inter-stream synchronization, with which Liger can manage the control dependencies between streams and eliminate the kernel launch overhead. Upon the hybrid approach, Liger designs the basic multi-stream scheduling algorithm.

Second, the multi-GPU architecture involves the resource contention problem when concurrently executing computation and communication kernels. Despite the significant differences in resource usage types, the contention [31] aroused in their parallel execution still leads to slow kernels, negating the benefits of overlapping and this degradation may disrupt the scheduling. Liger firstly adjusts the resource usage of communication kernels to mitigate the contention. Further, Liger takes a proactive approach that it profiles kernels with overlapped kernels executing concurrently and generates contention factors. By integrating contention factors into the scheduling algorithm, Liger anticipates the performance degradation, thereby preventing scheduling failures.

Third, the widely-varied execution durations of kernels in large model inferences can result in limited or no overlap. Kernels in different runs of large model inference vary in execution time. Since interrupting and resuming a kernel's execution is very difficult and expensive, each kernel continues execution until completion once started. Consequently, achieving an exact match between the execution time of computation and communication is rarely possible. Further, a lengthy kernel can block subsequent overlap as there is no available time slot that matches its duration. To increase the overlap extent, Liger dynamically decomposes a large kernel into more manageable units during runtime and achieves more accurate overlap. We make the following contributions:

- We identify the dilemma where balancing cost and effect under existing intra- and inter-operator parallelism in distributed large model inference (§2.2).
- We propose the novel interleaved parallelism, which can potentially achieve low latency at high throughput (§3.1).
- We present Liger, a distributed large model inference runtime system, that implements interleaved parallelism in the multi-GPU architecture.
- Liger ingeniously schedules computation and communication kernels across requests onto multiple CUDA streams by introducing the scheduling algorithm with the hybrid synchronization approach (§3.4), the resource contention anticipation approach (§3.5) and the runtime kernel decomposition approach(§3.6). To best of our knowledge, Liger is the first work that schedules inter-device kernels in a latency-critical scenario.
- We evaluate Liger and show its efficiency (§4).



(a) No direct interconnect. (b) Direct interconnect.

Figure 1. The illustration of multi-GPU architectures.

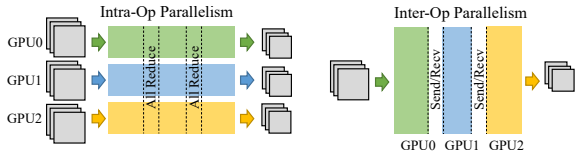


Figure 2. The illustration of existing parallelism.

2 Background and Motivation

In this section, we identify the dilemma where balancing cost and effect when applying existing parallelism for distributed large model inference. Next, we present challenges of jointly scheduling computation and communication kernels.

2.1 Multi-GPU Architectures

The multi-GPU architecture refers to a system setup where a node is equipped with more than one GPU. The architecture is widely used in deep learning training and is now being utilized in inference as models grow extremely large. CPUs handle complex or general tasks and delegate these highly repetitive processing tasks to the GPUs.

Fig. 1 shows multi-GPU architectures with two types of interconnects. In these architectures, the CPU sends commands to the GPU through a specific route, traversing from the CPU through the root complex, PCIe switch, PCIe interface, and finally reach the GPU. When performing collective communications between devices, the CPU sends the command to each GPU individually and GPUs perform collaboratively after all receiving the command, generally resulting in higher launch overhead. In the multi-GPU architecture with no direct interconnect, the communication between GPUs transfers via the PCIe switch, which only reaches a bandwidth of PCIe bus. Due to the significant demand on GPU-to-GPU communication, manufacturers incorporate direct communication links, such as NVLink [25] and AMD Infinity Fabric [1]. Although the CPU still controls GPUs through PCIe, GPUs transfer data to each other by NvLink and enables higher bandwidth and lower latency.

2.2 Dilemma in Distributed Large Model Inference

There are two parallelism approaches in making large model inference parallel, i.e. intra-operator parallelism [5, 35] and inter-operator parallelism [16, 18]. By distributing memory across multiple devices, they both allow the serving system to process larger models. However, they have different impacts on latency and throughput.

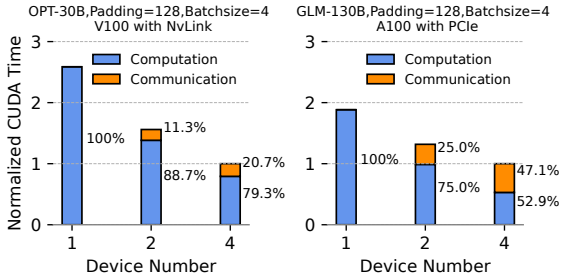


Figure 3. Strong scaling results of the intra-op. approach.

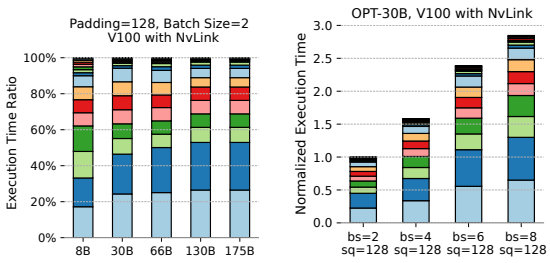
2.2.1 Intra-operator parallelism.

Fig. 2 illustrates the intra-operator parallelism approach. DL models include a series of operators, such as the matrix multiplication. Intra-operator approach partitions each operator across multiple devices, with each device executing a portion of the computation in parallel. Since each operator is executed by multiple devices simultaneously, it is capable to improve both latency and throughput. However, to eliminate data dependency between devices, it requires frequent collective communications throughout the execution, limiting the increase of throughput.

We deliver distributed inference case studies of two large language models, i.e. OPT-30B [41] and GLM-130B [40], on two multi-GPU architectures in FP16. OPT-30B is executed on a node with 4 NVIDIA V100 GPUs(16GB) connected via NVLink and the GLM-130B is performed on a node with 4 NVIDIA A100 GPUs(80GB) connected via PCIe. To show the strong scalability, we reduce the layer number of these models to make them accommodatable in less number of devices and report the theoretical results. Notably, each of these two models is stacked by layers with the same structure and reducing layer number will not impact the computational and communication features. In terms of OPT-30B on the V100 node, Fig. 3 shows that the overall execution time is reduced by 2.58× when increasing the device number from 1 to 4. Although the total execution time is largely reduced, the communication time takes 20.7% of the total time. Moving onto GLM-130B, which operates on a node with considerably lower bandwidth, the total execution time is only reduced by about 1.91× when increasing device number from 1 to 4, with communication consuming 47.1% of the total time. Thus, although the intra-operator approach is able to reduce the latency, it has significant resource waste, with computation units being left idle when communicating.

2.2.2 Inter-operator parallelism.

Fig. 2 demonstrates the approach of inter-operator parallelism. The target model is partitioned into disjoint stages and each stage is assigned to a specific device. Requests are processed in a pipeline [16, 18], with point-to-point communication occurring only every several layers to transfer intermediate results between



(a) Kernels in Different Models. (b) Kernels in varied input sizes.

Figure 4. Kernel durations in different models and inputs.

two devices. Unlike the intra-operator approach, the inter-operator approach cannot improve the latency and will slightly worsen it due to extra communications and load imbalance. Instead, when there are sufficient requests, it can improve the throughput by a factor nearly equal to the number of devices, largely exceeding that of the intra-operator parallelism.

Given the above, the intra-operator approach can largely reduce the latency at the expense of throughput, while the inter-operator approach can fully utilize hardware resources but doesn't improve latency. Thus, applying existing intra- and inter-approaches is stuck in a dilemma where balancing cost and effect.

2.3 Challenges

While interleaved parallelism shows promising results in addressing the dilemma, its implementation can be challenging and cumbersome due to challenges in GPU scheduling.

2.3.1 Communication Kernel Execution Lag. With the asynchronous kernel launch mode, the actual execution order on GPUs often differs from the order specified in the code, particularly with multiple streams. The experiments demonstrate that when computation kernels are executed concurrently, the startup of communication kernels is occasionally delayed. This is because there are multiple launch queues from the host to each GPU and the underlying scheduling logic in GPUs prioritize computation kernels. Due to the left-over scheduling policy that a new kernel is launched on the GPU only if there are required resources, the communication kernels will be delayed when meeting intensive computation kernels that occupies most compute units. Despite assigning communication kernels to the high-priority stream as suggested [22], this issue remains unresolved. Here the NCCL version is 2.15.1, which reports to honor stream priorities. To ensure the timely execution of communication kernels, it is possible to control the kernel launch based on the real-time GPU status. Nevertheless, such an approach will inevitably introduce overhead. Thus, it is challenge to effectively keep strict execution order of computation and communication kernels across multiple streams.

2.3.2 Hardware Resource Contention. Despite the differences in the major resource usage of communication and computation kernels, there indicates two sources of inefficiency [31]. (i) **Compute:** communication kernels should also utilize the compute resources of the GPU for collective updates and network driving. As a consequence, these compute-intensive kernels, such as general matrix multiplication, can severely impede the performance of each other when performed concurrently. (ii) **Memory Bandwidth:** both computation and communication need to write to and read from memory, resulting in interference. In this way, on the one hand, the contention problem will degrade the benefit, on the other hand, inaccurate kernel durations can lead to scheduling failures.

2.3.3 Widely-varied Kernel Duration. The widely-varied kernel durations in large model inference come from two aspects. First, each run of the large model inference has tens of different kernel types and they have great variance in durations. Fig. 4(a) presents the normalized duration of kernels in large language models on NVIDIA V100 GPUs, with parameter number ranging from 8 to 175 billions. As the model size increases, the variance in kernel duration becomes larger that few kernels take up most of the time. Second, the varying input sizes amplify the variance. Fig. 4(b) presents the normalized kernel durations in runs with different input sizes and kernel durations vary with inputs.

The widely-varied kernel duration problem results in limited or even no overlap. Since a kernel will continue execution until completion once started, it can be hard to identify two sets of kernels, one for computation and the other for communication, with exactly matched durations. Moreover, when there is a extremely lengthy kernel, it can block subsequent overlap, resulting in no overlap.

3 The Design of Liger

The key idea of Liger lies in the interleaved parallelism approach which can theoretically achieve low latency at high throughput. Basically, Liger targets the multi-GPU architecture of NVIDIA and achieves the interleaved parallelism through GPU scheduling techniques. It orchestrates computation and communication kernels of large model inference across multiple requests. We consider the scenario that Liger can be integrated into any serving architectures seamlessly as the runtime backend.

Fig. 5 demonstrates the overall design of Liger. After receiving requests and packing them as a batch in the serving system, the batch will be sent to Liger for further processing. The workflow of Liger features three primary phases: preprocessing phase, scheduling phase, and execution phase. Preprocessing phase includes an offline procedure and an online procedure. The offline procedure runs once for collecting runtime traces and contention factors before deployment. The online procedure assembles kernel launch functions of an inference as a list.

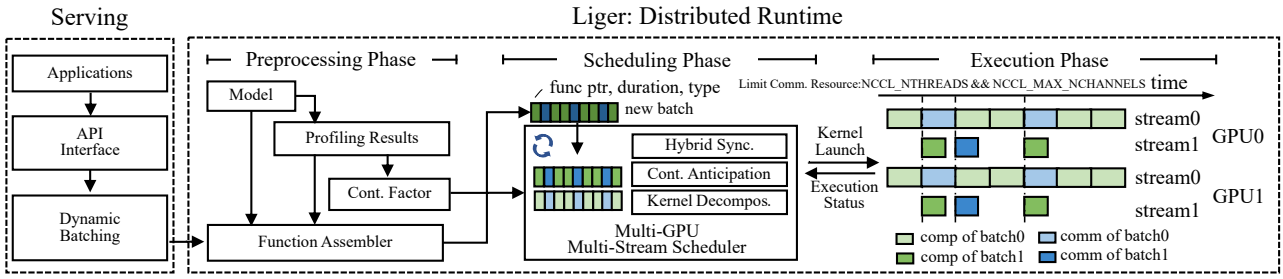


Figure 5. The System Overview of Liger.

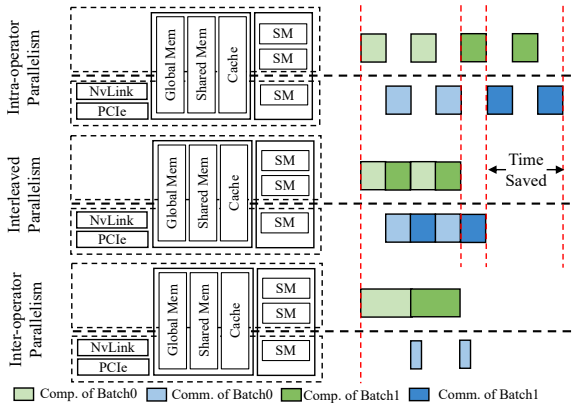


Figure 6. Kernel execution order in a single device of different parallelism approaches.

Then these lists will be handled by the scheduling phase. The scheduling algorithm periodically finds two kernel subsets with matched durations, one is computation and another one is communication, and launches them onto multiple streams of GPUs with a hybrid synchronization approach. To handle scheduling failures caused by the resource contention problem, Liger introduces the contention factor into the scheduling algorithm to anticipate the performance degradation. To increase the extent of overlap in the execution phase, the scheduler customizes the runtime kernel decomposition approach to tackle the widely-varied kernel duration problem. At last, these kernels will be interleaved as scheduled in the execution phase.

3.1 Interleaved Parallelism

Fig. 6 demonstrates the operator execution in a single device when employing the intra-operator, inter-operator (only the first pipeline stage), and interleaved parallelism. To understand the interleaved parallelism, it is necessary to divide the hardware resources of a device into two parts: computation resource and communication resource. Computation and communication kernels will share these resources and each kernel type mainly utilizes a specific type of resource.

For a specific model, the interleaved parallelism approach adopts the same partitioning strategy as the intra-operator parallelism approach. Consequently, this approach can distribute the workloads of a single operator to multiple devices and execute them concurrently. Moreover, when a new batch arrives, the interleaved parallelism approach parallelizes the inference of multiple batches, as in the inter-operator parallelism approach. Instead of placing different stages in different devices, the interleaved parallelism approach inserts the workloads of newly-arrived batches into the idle time of resources in another resource type, taking the idea of the time division multiplexing. For example, when the first batch starts to communicate, the interleaved parallelism approach will schedule computation kernels onto GPUs.

As a consequence, when requests arrive at a low rate, the interleaved parallelism degenerates to the intra-operator approach and achieves latency reduction. As the arrival rate gradually goes up, different batches start to overlap and the system can achieve a higher throughput.

3.2 Function Assembly

Liger generates a list of function wrappers for each newly-arrived batch. It utilizes information such as the batch size, sequence length, and the target model to assemble kernel launch functions and obtain their pointers. By executing these functions, Liger can launch corresponding GPU kernels of the inference. Additionally, each function wrapper encompasses not only the pointer of the kernel launch function but also details such as the kernel duration, the kernel type, the batch size, and the sequence length. These lists of function wrappers serve as a basis for making scheduling decisions. The function assembler in Liger also manages the execution status, such as memory management of intermediate results and recording the arrival order of batches.

3.3 Multi-GPU Multi-Stream Scheduler

Before delving into the specifics of how Liger schedules kernels onto GPUs, it is essential to outline several design principles guiding its approach:

- **Principle 1:** Ensuring the priority of the early-arrived batch is crucial to minimize its latency. It is essential to prevent kernels with the same type from interfering with the execution of the early-arrived batch when incorporating subsequent batch inferences.
- **Principle 2:** Ensuring the flexibility of the scheduler in order to handle any input sizes.

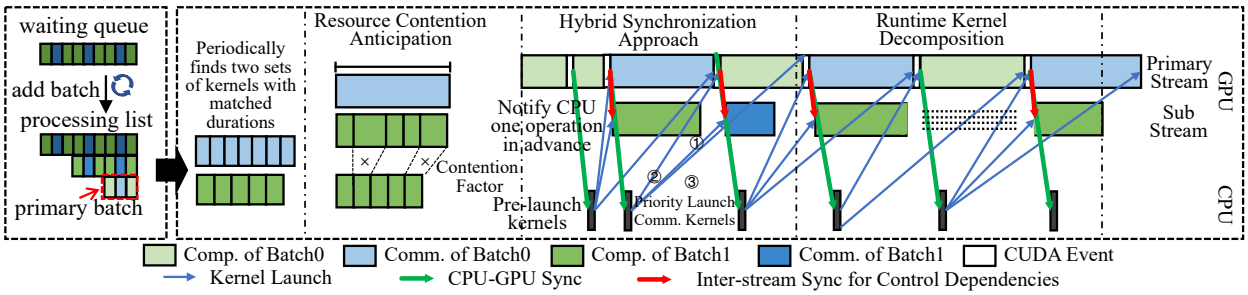


Figure 7. Scheduling computation and communication kernels across multiple batches onto multiple streams of GPUs.

- **Principle 3:** While adhering to the first and second principles, it should maximize the extent of overlap.

Fig. 7 demonstrates the multi-GPU multi-stream scheduler of Liger. The scheduler utilizes a waiting queue and a processing list to effectively manage incoming tasks. The waiting queue holds tasks that are pending execution, while the processing list contains a fixed number of tasks currently scheduled for processing. As tasks are completed and removed from the processing list, a new task is fetched from the waiting queue to take its place.

The scheduler periodically finds and launches two subsets of kernels in different types, from the primary batch and subsequent batches respectively. The length of primary subset is decided based on identifying the switch point, where the kernel type transitions between computation and communication. To accurately and efficiently place kernels of two subsets onto multiple streams, the scheduler designs the hybrid synchronization approach, which enables pre-launch to hide the kernel launch overhead and controls the execution order between streams without involving the CPU.

Next, since the contention problem may lead to slow kernels and disrupt the scheduling, besides reducing redundant resources allocated to communication kernels, the scheduler takes a proactive approach that considers the performance degradation in advance. This necessitates the cooperation of the profiling method and the scheduling algorithm.

At last, the scheduler handles the widely-varied kernel duration problem by dynamically decomposing lengthy kernels into fine-grained and manageable units during runtime. For kernels with long durations, specific decomposing strategies with equal capability are manually determined beforehand. When a kernel's duration exceeds the potential overlapping space, the scheduler will traverse potential strategies and find a suitable matching.

3.4 Scheduling Algorithm with the Hybrid Synchronization

Algorithm 1 presents the basic scheduling algorithm. The scheduler tends to identify two subsets of kernels with matched durations, one with communication kernels and another one with computation kernels, and launches them to two different streams for overlapping. To ensure the latency of the primary batch, as stated in **Principle 1**, the algorithm initially collects kernels from the primary batch until meeting the kernel type switch point and accumulates their durations. Then it traverses subsequent batches to identify kernels in another type with matched durations. With this algorithm, Liger can handle requests with various input sizes together, as stated in **Principle 2**. Further, Liger designs the hybrid synchronization approach to avoid kernel startup delay and ensure strict overlap between computation and communication kernels.

Algorithm 1: The Basic Scheduling Algorithm

```
   Data: FuncVec, type, time ▷ list, kernel type, duration
 1 while True do
 2     FuncVec ← update_list(), time = 0 ;
 3     ▷ collect kernels from the primary batch
 4     while !FuncVec[0].empty() do
 5         if FuncVec[0].switch() then
 6             ▷ next kernel type switches
                time += FuncVec[0].duration;
                type = FuncVec[0].type;
                SubSet0.pushback(FuncVec[0].pop()) ;
 7             break;
 8         else
 9             ▷ next kernel type the same
                time += FuncVec[0].duration;
                SubSet0.pushback(FuncVec[0].pop());
10     ▷ collect kernels from subsequent batches
11     for V in FuncVec[1:] do
12         while time > 0 do
13             if V.is_not_same(type) then
14                 if time > V.duration then
15                     time = 0;
16                 else
17                     time -= V.duration;
18                     SubSet1.pushback(V.pop());
19             else
20                 break;
21     SubSet0.launch(Stream0), SubSet1.launch(Stream1) ;
```

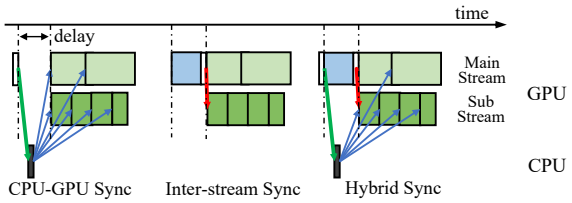


Figure 8. Synchronization approaches to coordinate kernel execution order in multiple CUDA streams.

The hybrid synchronization approach features pre-launching and inter-stream control. The CUDA API offers two standard synchronization, namely CPU-GPU synchronization and inter-stream synchronization. As shown in Fig. 8, the CPU-GPU synchronization allows for coordination between the CPU and GPU. Once the CPU is informed that all kernels have terminated, it can launch the communication subset first and then the computation subset for achieving the precise control. However, this synchronization will suffer from the kernel launch overhead and damage the performance.

Moving onto the inter-stream synchronization, as illustrated in Fig. 8, this approach achieves coordination between CUDA streams without involving the CPU. Instead, the inter-stream synchronization controls the execution order of different streams by utilizing the inter-stream communication capability. Liger can theoretically use it to control the kernel execution order across streams throughout the entire working process. Although Fig. 8 explicitly presents the CUDA event as a white block, it introduces very limited overhead in this approach. Unfortunately, our experiments indicate that launching all kernels and controlling their execution order solely by inter-stream synchronization can lead to lag problems when meeting communication kernels.

As shown in Fig. 8, the hybrid synchronization approach inserts two CUDA events before and after the kernel whose next kernel switches to another type. When the first CUDA event is triggered, it notifies the CPU to launch these two subsets of kernels, with the communication subset ranking first. Here **CUDA_DEVICE_MAX_CONNECTIONS** is set to 2. During the kernel launching process, there is still one kernel running in the GPU, being able to hide the kernel launch overhead. These newly-launched kernels will not be executed immediately. Instead, their execution begins until the second CUDA event is triggered. This event takes the inter-stream synchronization, directly communicating between streams without involving the CPU. Thus, Liger achieves precise control of kernel execution order without suffering from kernel launch overhead.

3.5 Contention Mitigation and Anticipation

Liger requires to profile durations of kernels for scheduling in Algorithm 1. However, the contention problem will make the kernel duration inaccurate and scheduling with inaccurate values can result in concurrent execution of kernels with the same type, causing more serious contention. Liger handles the contention problem from two aspects: (i) manage resource allocation of kernels and (ii) anticipate the performance degradation in the scheduling algorithm.

Liger mitigates the contention problem through reducing the redundant resource allocation of communication kernels. Specifically, in the CUDA programming model, the resource requirements of a kernel are determined when programming and cannot be controlled afterwards. Since there are many different types of computation kernels and only several collective communication kernels are used, we choose to inspect collective kernels of NCCL library. We find that NCCL generally allocates redundant CUDA blocks to communication kernels by default and less blocks are enough to saturate the peak bandwidth. In this way, Liger reduces the number of CUDA blocks and CUDA threads used for communication by using environment variables **NCCL_NTHREADS** and **NCCL_MAX_NCHANNELS**.

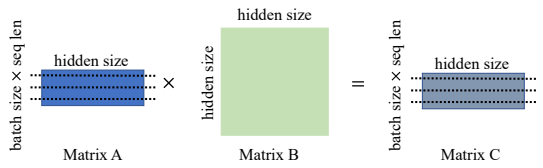
Next, Liger anticipates the resource contention problem and ensures a correct scheduling by incorporating contention factors into Algorithm 1. The conventional profiling method collects kernel durations under no-load conditions and one direct idea is to collect all possible durations of a kernel during concurrent execution. However, this approach becomes impractical due to two main factors: (i) two kernels do not run concurrently throughout the entire process and (ii) the unacceptable search space.

By contrast, the contention factor strategy does not profile all possible durations. First, it only focuses on lengthy computation kernels with intensive computation and communication kernels. Second, it takes the concurrent profiling of these kernels with different inputs and determines maximum contention factors between them. Third, Liger still uses the standard profiling results and only scales kernel durations of subsequent batches with contention factors when scheduling. In this way, at the expense of the overlapping extent, its accumulated duration will never exceed that of the primary subset and guarantee **Principle 1**.

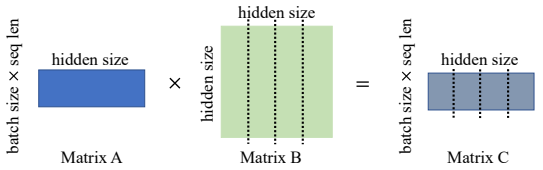
3.6 Runtime Kernel Decomposition

When durations of kernels across batches vary significantly, the scheduler struggles to find matched durations of computation and communication kernels, making the interleaved parallelism approach inefficient. Liger addresses this by breaking down lengthy kernels into multiple fine-grained and more manageable kernels with equal capabilities.

The kernel decomposition is a manual decision-making process in Liger. In general, giant kernels of large models primarily include collective communication kernels and GEMM kernels. We adopt the equal division strategy for them and profile durations of all possible division within a specific number of divisions. For instance, setting the division factor as 8, we profile durations for divisions ranging from 1/8 to



(a) horizontal decomposition



(b) vertical decomposition.

Figure 9. Matrix multiplication kernel decomposition.

7/8. With the profiling results, the scheduler can dynamically determine the kernel decomposition strategy at runtime.

Different decomposition strategies will greatly affect the performance, especially for GEMM kernels. Fig. 9 displays strategies for decomposing GEMM kernels in large language models. The horizontal approach will experience a notable reduction in computation intensity, with the accumulated duration of all fine-grained kernels significantly exceeding that of the original kernel. In comparison, the vertical approach performs much better. This comes from two factors. First, the matrix A is already a skinny matrix, dividing it horizontally would make it even skinnier, resulting in worse data locality. Second, the matrix B is larger than matrix A. When dividing matrix B, the amount of memory I/O required is much less. Thus, it is essential for Liger to carefully configure the GEMM decomposition. Through this approach, Liger can achieve a higher degree of overlap, maximizing the system's overall performance and enhancing the efficiency.

4 Implementation and Evaluation

We implement a prototype system for Liger with about 3k lines of code in C++. Since we take large language models as the serving target, we reuse CUDA kernels mainly from a highly optimized transformer inference system, FasterTransformer [23], to build models. In this section, we evaluate the serving ability of Liger prototype under a variety of models, nodes, and tasks.

4.1 Experimental Setup

Node testbed. We deploy Liger on two multi-GPU nodes. One has 4 NVIDIA Tesla V100 GPUs (16GB) with NVLink. The NCCL-tests [24] report a peak all-reduce bandwidth at 32.75 GB/s. Another one has 4 NVIDIA Ampere A100 GPUs (80GB) and they communicate through PCIe switch with a peak all-reduce bandwidth at 14.88 GB/s. Specifically, we use gcc~8.4.0, CUDA~11.3, MVAPICH~3.2.1, and NCCL~2.18.5.

Table 1. Model Specifications.

Name	Parameters	Layers	Heads	Hidden Size	Prec.
OPT-30B	60GB	48	56	7168	FP16
OPT-66B	132GB	64	72	9216	FP16
GLM-130B	260GB	70	96	12288	FP16

Model setup. Since Transformer [36] is the most common backbone for large models, we choose three large Transformer models with different model sizes. As detailed in Table 1, they are OPT-30B, OPT-66B [41], and GLM-130B [40]. GLM-130B has the same layer setup as GPT-3.

Metrics. We use both the average latency and throughput as the evaluation metrics. Since a job may not be executed immediately upon arrival, the latency includes two parts, i.e. the pending time and the cuda execution time. As for throughput, it refers to the number of requests a system can handle within a given time. Since latency and throughput are mutually exclusive in some extend, to demonstrate advantages of Liger under various real-world workloads, we constantly increase the arrival rate of requests and conduct several sets of experiments under different arrival rates. We collect these metrics from continuously serving 2000 requests.

Baseline Methods. We implement and compare Liger with existing intra- and inter-operator parallelism approaches used in state-of-the-art inference systems [2, 7, 23, 28, 39]:

- **Intra-operator Parallelism (Intra-Op)**: The intra-operator parallelism approach from Megatron-LM [35], that splits the weight tensor in transformer models to parallelize the GEMM operators. Two All-reduce synchronizations are required in each transformer layer.
- **Inter-operator Parallelism (Inter-Op)**: The inter-operator parallelism approach [16] partitions transformer models into equal stages, with each stage residing on a separate device. It necessitates only a single point-to-point communication between stages.
- **Theoretical Inter-operator Parallelism (Inter-Th)**: Considering that the duration of multiple partitioned kernels often differs from that of the original kernel, primarily due to the kernel implementation, the theoretical inter-operator parallelism uses partitioned kernels which are taken from the intra-operator parallelism approach.

4.2 General Performance Evaluation

In the section, we compare Liger against baseline methods on randomly generated traces with sequence lengths ranging from 16 to 128. In large model scenarios, a small batch size can already saturate the GPU and the CUDA execution latency grows linearly with larger batch size [33]. Thus, we use small batch sizes of 2, 4, and 8. All-reduce and GEMM kernels are equally decomposed with a factor of 8. Given the memory constraint, we perform the OPT-30B model on the V100 node and all models on the A100 node. Detailed results are presented in Fig. 10.

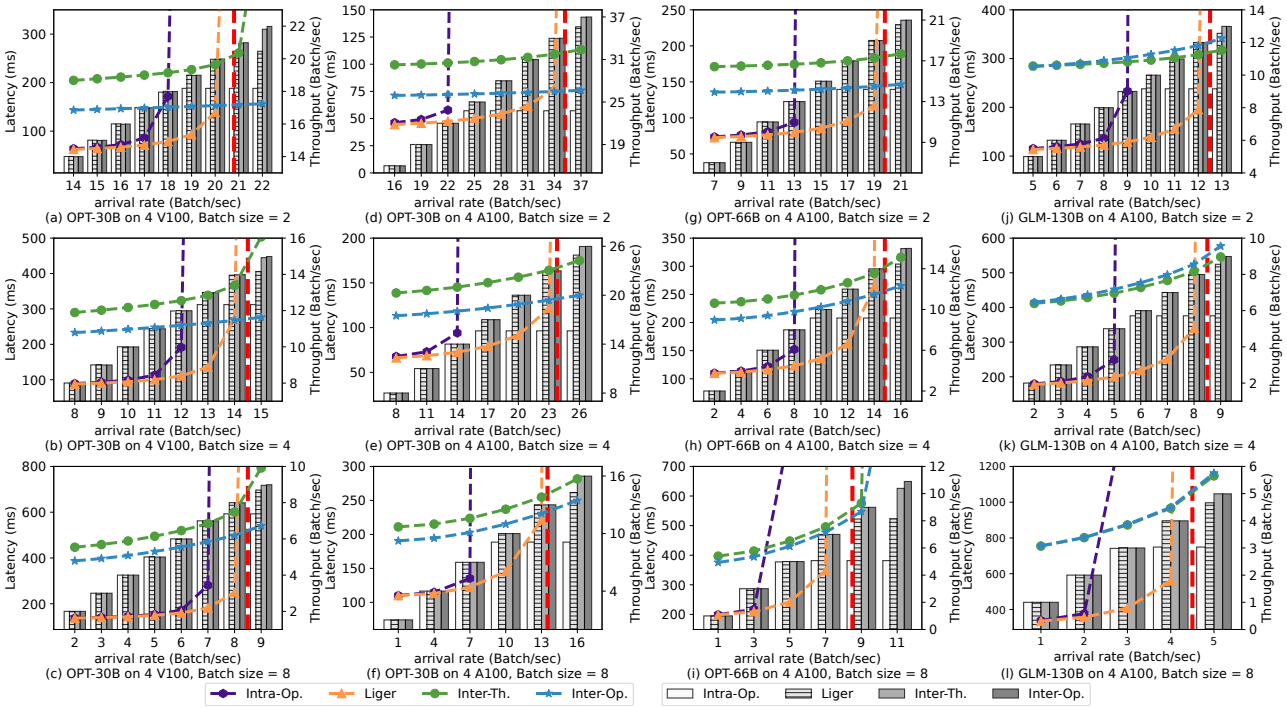


Figure 10. Latency and throughput results as request arrival rate increases with randomly generated traces. Vertical red lines represent that the arrival rate starts to exceed Liger's throughput. The latency is in line and the throughput is in bar.

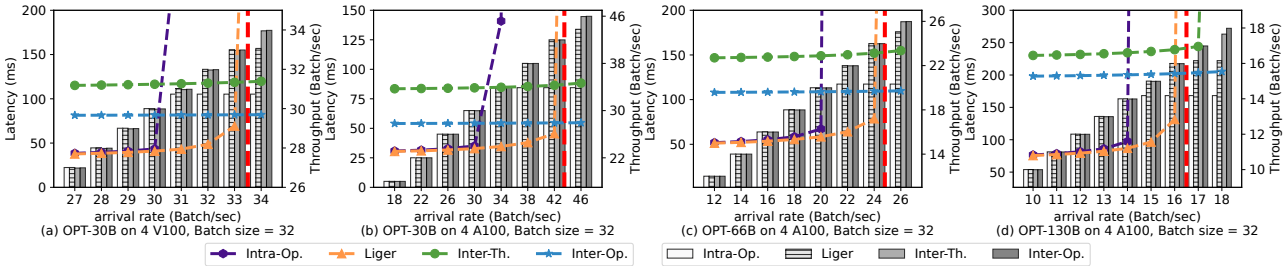


Figure 11. The latency and throughput results of generative task. The sequence length of the starting point is 16.

In all cases shown in Fig. 10, Liger presents significant advantages in both latency and throughput, specifically before the arrival rate exceeds the throughput of Liger (red lines). When the arrival rate is low, the interleaved parallelism functions similarly to the intra-op parallelism. The interleaved parallelism has the same latency with the intra-operator approach, which is much better than that of the inter-operator parallelism. As the arrival rate increases and exceeds the throughput of the Intra-Op, Liger can continue to process these batches, reaching even higher throughput. Since we use a constant request rate instead of a fluctuated request rate, our approach simultaneously advances over the best of intra- and inter-operator approaches in a relatively narrow arrival rate window. Particularly, the throughput can be increased by an average of 1.15× in the V100 node, and 1.52× in the A100 node, when comparing to the Intra-Op. Meanwhile, before exceeding the upper limit of Liger's throughput, Liger can still maintain significant advantages in the latency when comparing to the inter-operator approach. Basically, Liger reduces the latency by an average of 45.4% and 59.1% in the V100 node, and 35.8% and 42.2% in the A100 node, when comparing to Inter-Op and Inter-Th. Thus, it is certified that Liger can achieve low latency at high throughput compared to existing intra- and inter-parallelism approaches.

The effectiveness of Liger is heavily determined by the interconnect environment. Based on the OPT-30B's performance on V100 and A100 nodes, the throughput improvement of Liger on the A100 node is much more obvious. This is because the weaker interconnect on the A100 node results in more communication durations, allowing Liger to leverage interleaved parallelism and benefit more from it. Regarding contention factors, the V100 node uses a scaling factor of 1.1 for scaling the accumulated duration of subsets, while the A100 node uses a scaling factor of 1.15. It may seem counterintuitive that the A100 node requires a larger factor, even though it possesses more computing resources but has weaker interconnect bandwidth. However, it's important to note that both the primary and secondary queues

are affected by hardware contention. The contention factor is primarily used to ensure that the execution time of the secondary queue does not exceed that of the primary queue.

Next, batch size and inherent model features can largely impact Liger's effectiveness. In Fig. 10, the vertical comparison presents the performance with different batch sizes and the horizontal comparison presents the performance with different model sizes. As batch size increases, Liger achieves the better throughput improvement. This is because a larger batch size allows computation kernels to better saturate GPUs and the communication relatively increases, leaving more room for Liger to overlap. In comparison, the model size shows an opposite impact on the throughput improvement that a larger model size benefits less from applying Liger. This is because the computation amount of these large language models exhibit a quadratic relationship with their size, whereas the communication amount is a linear relationship. Thus, a larger model has relatively less communication.

In addition, Fig. 10(j)(k) reveals unusual results where Inter-Th exhibits the better throughput than Inter-Op. Profiling results indicate that the accumulated duration of four kernels partitioned in Intra-Op is shorter than that of a single kernel in Inter-Op, related to the GEMM implementation.

4.3 Performance Evaluation on Generative Tasks

Generative tasks are important workloads in large model inference, involving two distinct execution phases: the initial conditioning phase and the incremental sampling phase. The initial conditioning phase performs similarly to general tasks in §4.2 that takes a sentence as input and outputs the encoded representation as the starting point for generating a response. The incremental sampling phase is unique that it generates one token at a time, gradually forming a complete sentence. This section focuses on the incremental sampling phase, employing the KV cache technique [28]. We perform one iteration of the sampling phase constantly with a sequence length of 16 as the starting point and a batch size of 32.

As shown in Fig. 11, it is evident that Liger yields improvements in both latency and throughput, while the effect is relatively weaker. Specifically, compared to Intra-Op, Liger can enhance the throughput by up to 1.08×, 1.29×, 1.23×, and 1.13× respectively in these evaluations, while also achieving better latency. Compared to Inter-Op and Inter-TH, before saturating Liger's throughput, it can reduce the average latency. However, due to the lower computational intensity of generative tasks, the relatively less communication makes Liger less room to take effect.

4.4 Strong Scalability Analysis

We conduct strong scaling experiments of serving OPT-30B on 1, 2, and 4 A100 GPUs. In Fig. 12, latency and throughput results are selected from data points as the arrival rate constantly increases. It shows that Liger can improve latency and throughput as the number of GPUs increases. Moreover,

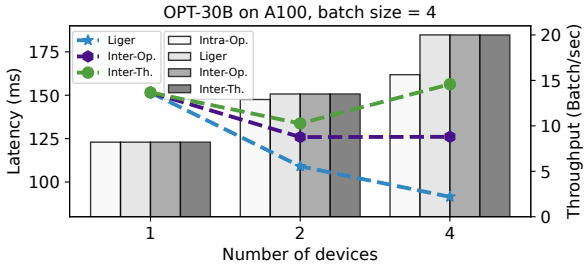


Figure 12. The strong scaling results.

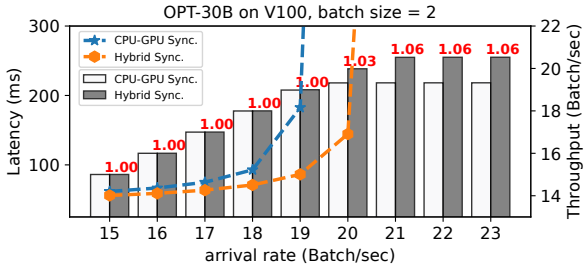


Figure 13. The performance wi/wo pre-launching.

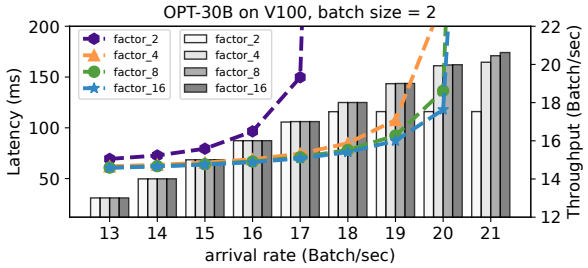


Figure 14. The performance under different division factor.

Liger outperforms the intra-operator approach in throughput and the inter-operator approach in latency. It is worth noting that the impact of Liger is less pronounced in the 2-GPU configuration due to a lower communication ratio.

4.5 Benefits of Hybrid Synchronization

This section presents the benefits of the hybrid synchronization approach. We integrate Liger with the hybrid synchronization approach or the CPU-GPU synchronization approach and compare them. The evaluation is conducted on the V100 node, and serves OPT-30B with a batch size of 2. In Fig. 13, it is observed that Liger, when paired only with the CPU-GPU synchronization approach, performs unfavorably compared to the hybrid synchronization approach, with an obvious performance drop on both latency and throughput. The kernel launch overhead in Liger is more pronounced compared to a single GPU program. When we profile null kernels, the launch delay is around 5 microseconds, while in Liger, it requires all communication kernels in multiple GPUs to complete before proceeding, the launch overhead can be significantly longer, exceeding 20 microseconds. This can be explained from two aspects: 1) inconsistent launching time between GPUs, and 2) PCIe contention exacerbates the inconsistency.

4.6 Benefits of Kernel Decomposition

To explore how the decomposition granularity impacts Liger, we setup experiments (Fig. 14) with decomposition factor of 2, 4, 8, and 16. The evaluation is on the V100 node, and we serve OPT-30B with a batch size of 2.

Liger demonstrates better latency and throughput results with a larger decomposition factor. This is because a larger decomposition factor can result in a more precise decomposition granularity and the scheduler can identify two subsets of kernels with more closely matched durations. This can also be explained that the decomposing results of a larger factor can encompass all possible decomposition of a small factor. In addition, since too small a kernel cannot saturate GPUs and these small kernels have similar durations, the benefits of enlarging the decomposition factor gradually decrease.

5 Related Work

Runtime System for Model Inference. Advances in deep learning have promoted a proliferation of model inference runtime systems. There includes general-purpose systems, like TensorRT [26] and ONNX [6], and systems for a specific set of models, such as for transformers [11, 23, 45]. For large models, inference systems [7, 23, 29] take model parallelism from training systems and solve a unique set of constraints. Also, there are runtime systems optimized for specific tasks, such as for generative tasks [28, 37, 39, 43] and for throughput-oriented scenario [15, 34] using the offloading strategy. Liger targets the latency-critical scenario and is mostly orthogonal to these works.

GPU Scheduling and Multiplexing. There has been a large amount of prior work focusing on improving the utilization of GPUs by co-location. TimeGraph [17] and GPUSync [10] guarantee the performance of real-time kernels by assigning different priorities in GPU multiplexing. Liger fails in scheduling communication kernels with the idea of priority. Moving onto DL-related works in GPU scheduling, AntMan [38] co-locates multiple DL training jobs and PilotFish [42] co-locates DL training with cloud gaming. Besides throughput-oriented GPU scheduling works, REEF [14] co-locates multiple inference tasks and proposes a fast kernel preemption method to ensure the performance of latency-critical tasks. AlpaServe [19] focuses on multiplexing GPUs in a cluster and efficiently serving multiple large models.

Model Parallelism for Large Model Training. Due to the memory constraint, model Parallelism is used to train large models that cannot fit into a single GPU. There are two major sets of model parallelism approaches, namely intra-operator parallelism [4, 30, 32, 35] and inter-operator parallelism [16, 20, 21]. Besides, MPress [44] and Mobius [12] optimize the model parallelism training for the hardware architecture with limited communication bandwidth. Liger is largely orthogonal to these approaches.

6 Conclusion

This paper introduces Liger, a distributed large model inference runtime system featuring the innovative interleaved parallelism. Liger incorporates the scheduling algorithm with the hybrid synchronization approach, resource contention anticipation approach, and runtime kernel decomposition approach to dynamically schedule computation and communication kernels in the multi-GPU architecture. Our evaluations show that Liger can achieve low latency at high throughput, to some extent, tackling the current dilemma where balancing cost and effect. In a specific 4-device case, Liger reduces the average latency by 36.0% while maintaining the same throughput compared to the inter-operator approach. Meanwhile, it improves the throughput by 1.34× with improved average latency compared to the intra-operator approach.

7 Acknowledgments

This research was supported by the National Key R&D Program of China 2021YFB0301300, and also received support from programs: the Major Program of Guangdong Basic and Applied Research: 2019B030302002, the Guangdong Province Special Support Program for Cultivating High-Level Talents: 2021TQ06X160, the Pazhou Lab Research Project: PZL2023KF0001, and the CCF-Baidu Open Fund: CCF-BAIDU202302.

References

[1] AMD. 2023. *AMD Infinity Architecture.* https://www.amd.com/en/technologies/infinity-architecture

[2] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, et al. 2022. DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC).* IEEE Computer Society, 646–660.

[3] Tom Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Shenggan Cheng, Ziming Liu, Jiangsu Du, and Yang You. 2023. ATP: Adaptive Tensor Parallelism for Foundation Models. *arXiv preprint arXiv:2301.08658* (2023).

[5] Shenggan Cheng, Xuanlei Zhao, Guangyang Lu, Jiarui Fang, Zhongming Yu, Tian Zheng, Ruidong Wu, Xiwen Zhang, Jian Peng, and Yang You. 2023. FastFold: Reducing AlphaFold Training Time from 11 Days to 67 Hours. arXiv:2203.00854 [cs.LG]

[6] ONNX Runtime developers. 2018. *ONNX Runtime.* https://github.com/microsoft/onnxruntime

[7] Jiangsu Du, Ziming Liu, Jiarui Fang, Shenggui Li, Yongbin Li, Yutong Lu, and Yang You. 2022. EnergonAI: An Inference System for 10-100 Billion Parameter Transformer Models. *arXiv preprint arXiv:2209.02341* (2022).

[8] Mengnan Du, Subhabrata Mukherjee, Yu Cheng, Milad Shokouhi, Xia Hu, and Ahmed Hassan Awadallah. 2021. What do compressed large language models forget? robustness challenges in model compression. *arXiv e-prints* (2021), arXiv–2110.

[9] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1:*

Long Papers). 320–335.

[10] Glenn A Elliott, Bryan C Ward, and James H Anderson. 2013. GPUSync: A framework for real-time GPU management. In *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 33–44.

[11] Jiarui Fang, Yang Yu, Chengduo Zhao, and Jie Zhou. 2021. Turbotransformers: an efficient gpu serving system for transformer models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 389–402.

[12] Yangyang Feng, Minhui Xie, Zijie Tian, Shuo Wang, Youyou Lu, and Jiwu Shu. 2023. Mobius: Fine tuning large-scale models on commodity gpu servers. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 489–501.

[13] Github. 2023. *Copilot*. https://github.com/features/copilot

[14] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 539–558.

[15] Xu Han, Guoyang Zeng, Weilin Zhao, Zhiyuan Liu, Zhengyan Zhang, Jie Zhou, Jun Zhang, Jia Chao, and Maosong Sun. 2022. BMInf: An efficient toolkit for big model inference and tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 224–230.

[16] Yanping Huang, Youlong Cheng, Ankur Bapna, et al. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 103–112.

[17] Shinpei Kato, Karthik Lakshmanan, et al. 2011. TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*.

[18] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, et al. 2020. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *International Conference on Learning Representations*.

[19] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 663–679.

[20] 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*. PMLR, 6543–6552.

[21] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 1–15.

[22] NVIDIA. 2023. *Fast Multi-GPU collectives with NCCL*. https://developer.nvidia.com/blog/fast-multi-gpu-collectives-nccl/

[23] NVIDIA. 2023. *FasterTransformer*. https://github.com/NVIDIA/FasterTransformer

[24] NVIDIA. 2023. *NCCL Tests*. https://github.com/NVIDIA/nccl-tests/tree/master

[25] NVIDIA. 2023. *NVLink and NVSwitch*. https://www.nvidia.com/en-us/data-center/nvlink/

[26] NVIDIA. 2023. *TensorRT*. https://github.com/NVIDIA/TensorRT

[27] OpenAI. 2023. *Chatgpt*. https://chat.openai.com/chat

[28] Reiner Pope, Sholto Douglas, et al. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023).

[29] Samyam Rajbhandari, Conglong Li, and et al. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning*. PMLR, 18332–18346.

[30] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[31] Saeed Rashidi, Matthew Denton, Srinivas Sridharan, Sudarshan Srinivasan, Amoghavarsha Suresh, Jade Nie, and Tushar Krishna. 2021. Enabling compute-communication overlap in distributed deep learning training platforms. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 540–553.

[32] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems* 31 (2018).

[33] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[34] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. (2023).

[35] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, et al. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2019). arXiv:1909.08053 http://arxiv.org/abs/1909.08053

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[37] Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Fast Distributed Inference Serving for Large Language Models. *arXiv preprint arXiv:2305.05920* (2023).

[38] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. {AntMan}: Dynamic scaling on {GPU} clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 533–548.

[39] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for {Transformer-Based} Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538.

[40] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414* (2022).

[41] Susan Zhang, Stephen Roller, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[42] Wei Zhang, Binghao Chen, and et al. 2022. {PilotFish}: Harvesting Free Cycles of Cloud Gaming with Deep Learning Training. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 217–232.

[43] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, et al. 2023. H_2O : Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *arXiv preprint arXiv:2306.14048* (2023).

[44] Quan Zhou, Haiquan Wang, and et al. 2023. MPress: Democratizing Billion-Scale Model Training on Multi-GPU Servers via Memory-Saving Inter-Operator Parallelism. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 556–569.

[45] Zhe Zhou, Xuechao Wei, Jiejing Zhang, and Guangyu Sun. 2022. {PetS}: A Unified Framework for {Parameter-Efficient} Transformers Serving. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 489–504.

A Artifact Description

A.1 Availability

The artifact of Liger runtime system is available at: https://figshare.com/articles/thesis/Liger_Interleaving_Intra-_and_Inter-Operator_Parallelism_for_Distributed_Large_Model_Inference/24902067

B Requirements

Libraries: The artifact depends on MPI, CUDA, NCCL nlohmann -json, and these versions of libraries are passed in our experiments:

- MVAPICH2@2.3.7, @3.2.1.
- CUDA@11.3@12.2
- NCCL@2.15.5, @2.18.5-1
- nlohmann-json@3.11.2.

Hardware: The artifact requires a NVIDIA multi-GPU node. It has been tested on nodes equipped with 4 Tesla V100 GPUs (the 1st generation of NVLINK) or 4 Ampere A100 (PCIe) GPUs.

C Running Experiments

```
make
export NCCL_MAX_NCHANNELS=3
export CUDA_DEVICE_MAX_CONNECTIONS=2
mpirun -np2 ./main
```

D Some details of the Artifact

- You can configure the model, adjust the request rate, or the decomposition factor in main.cu. The default configuration comes from GLM-130B.
- The process of function assembly can be found at src/layer/bert.h, it adopts an iterative approach to store kernel launch functions in a vector (§3.2).
- The scheduling algorithm is implemented in run_stream_sync.cpp. You can also find the hybrid synchronization method and the contention factor here (§3.4).
- We manage the kernel decomposition in src/init.h/Sample (§3.6).
- Since nodes vary in computation and communication ability, it is necessary to specify the arrival rate for your node and there exists an arrival rate range where Liger performs better than both intra- and inter-operator parallelism approaches.