LIGHTSEQ: Sequence Level Parallelism for Distributed Training of Long Context Transformers

Dacheng Li [*,†] **Rulin Shao** [*,‡] **Anze Xie** [§] **Eric P. Xing** [†,†]

Joseph E. Gonzalez [†] **Ion Stoica** [†] **Xuezhe Ma** [**] **Hao Zhang** [§]

[b] UC Berkeley [w] University of Washington [s] UCSD [c] CMU [m] MBZUAI [u] USC

Abstract

Increasing the context length of large language models (LLMs) unlocks fundamentally new capabilities, but also significantly increases the memory footprints of training. Previous model-parallel systems such as Megatron-LM partition and compute different attention heads in parallel, resulting in large communication volumes, so they cannot scale beyond the number of attention heads, thereby hindering its adoption. In this paper, we introduce a new approach, LIGHTSEQ, for long-context LLMs training. LIGHTSEQ has many notable advantages. First, LIGHTSEQ partitions over the sequence dimension, hence is agnostic to model architectures and readily applicable for models with varying numbers of attention heads, such as Multi-Head, Multi-Query and Grouped-Query attention. Second, LIGHTSEQ not only requires up to 4.7× less communication than Megatron-LM on popular LLMs but also overlaps the communication with computation. To further reduce the training time, LIGHTSEQ features a novel gradient checkpointing scheme to bypass an forward computation for memory-efficient attention. We evaluate LIGHTSEQ on Llama-7B and its variants with sequence lengths from 32K to 512K. Through comprehensive experiments on single and cross-node training, we show that LIGHTSEQ achieves up to 1.24-2.01× end-to-end speedup, and a 2-8× longer sequence length on models with fewer heads, compared to Megatron-LM. Code is available at `https://github.com/RulinShao/LightSeq`.

1 Introduction

Transformers with long-context capabilities have enabled fundamentally new applications, such as comprehensive document understanding, generating a complete codebase, and extended interactive chatting (Osika, 2023; Liu et al., 2023; Li et al., 2023). However, training LLMs with long sequences induces large activation memory footprints, posing new challenges to existing distributed systems.

One effective method for reducing these large activation memory footprints is to partition the activation across devices. To achieve this, existing systems like Megatron-LM (Korthikanti et al., 2023; Shoeybi et al., 2019) usually partition the attention heads. However, this design poses a strong assumption that the number of attention heads must be divisible by the parallelism degree, which does not hold for many model architectures. For example, Llama-33B has 52 attention heads, which is not divisible by commonly chosen parallelism degrees such as 8, 16, and 32, according to the topology

[*]Authors contributed equally.

Workshop on Advancing Neural Network Training at 37th Conference on Neural Information Processing Systems (WANT@NeurIPS 2023).

of NVIDIA clusters. In addition, partitioning attention heads restricts the maximum parallelism degree to be no greater than the number of attention heads. However, many popular LLMs do not have enough attention heads for it to scale up, e.g., CodeGen (Nijkamp et al., 2022) only has 16 attention heads. Moreover, many works have shown that the future Transformer architecture design may have even fewer attention heads. For example, Bian et al. (2021) demonstrates that Transformers with a single head outperforms its multi-head counterparts, representing a challenging scenario for solutions like Megatron-LM. To scale beyond the number of heads, we propose partitioning solely the input tokens (i.e., sequence parallelism) rather than the attention heads. We present a solution that is agnostic to the model architecture and exhibits a maximal parallelism degree that scales with the sequence length. Specifically, we introduce a parallelizable and memory-efficient exact attention mechanism, DISTATTN, in (§3.1). Our design enables opportunities for overlapping, where we can hide communication into attention computation(§ 3.2). We also propose a load-balancing technique to avoid the computation bubble caused by the unbalanced workload in causal language modeling (§3.2). While extending the FlashAttention (Dao, 2023) algorithm to DISTATTN, we found a way to leverage the underlying rematerialization logic to si gnificantly improve the speed of gradient checkpointing training (§ 3.3). This technique also applies to non-distributed usage of memory-efficient attention, and in our experiments translates to an additional 1.31× speedup (§ 4.3).

Our main contributions are:

1. We design LIGHTSEQ, a long-context LLM training prototype based on sequence-level parallelism. We develop a distributed memory-efficient exact attention DISTATTN, with novel load balancing and communication overlapping scheduling for causal language modeling.
2. We propose a novel checkpointing strategy that bypasses one attention forward pass when using memory-efficient attention with gradient checkpointing training.
3. We evaluate LIGHTSEQ on Llama-7B and its variants with different attention heads patterns, and demonstrate up to 2.01× end-to-end speedup compared to Megatron-LM in long-context training. We further show that LIGHTSEQ scales beyond the number of attention heads and enables 2-8× longer sequences training.

2 Related work

Memory-efficient attention. Dao et al. (2022) and Lefaudeux et al. (2022) propose to use an online normalizer (Milakov & Gimelshein, 2018) to compute the attention in a blockwise and memory-efficient way. It reduces peak memory usage by not materializing large intermediate states, e.g. the attention matrix or the up projection matrix output of the MLP layers (Liu & Abbeel, 2023). Instead, the attentions are computed in smaller blocks and only the final activation are stored. In the backward pass, the intermediate states need to be recomputed. Research on sparse attention computes only a sparse subset of the attention score, which also reduces the memory footprints yet may lead to inferior performance (Beltagy et al., 2020; Sun et al., 2022; Zaheer et al., 2020). In this work, we limit our scope to exact attention.

Sequence parallelism, model parallelism, and FSDP. Li et al. (2021) is among the first to parallelize along the sequence dimension. However, it is not optimized for the computational pattern of causal language modeling and is incompatible with memory-efficient attention, which are crucial to long-context LLM training. Model parallelism partitions model parameters and also distributes the activation in parallel LLM training. Megatron-LM (Korthikanti et al., 2023) proposes a hybrid usage of tensor parallelism and sequence parallelism to better reduce the activation on a single device and is the main baseline of the paper. Fully sharded data-parallelism (FSDP) (Zhao et al., 2023; Rajbhandari et al., 2020) distributes optimizer states, gradients, and model parameters onto different devices and gathers them on-the-fly. It is orthogonal to our work, and we use LIGHTSEQ in tandem with FSDP to further reduce memory acquired by models in experiments.

Gradient checkpointing. Gradient checkpointing (Chen et al., 2016) trades computation for memory by not storing the activation for certain layers and recomputing their activations during forward. Selective checkpointing (Korthikanti et al., 2023) proposes to only recompute the attention module as it requires large memory but with small FLOPs (in smaller context length). Checkmate (Jain et al., 2020) searches optimal checkpointing using integer linear programming. However, none of these designs have considered memory-efficient attention kernels which perform recomputation

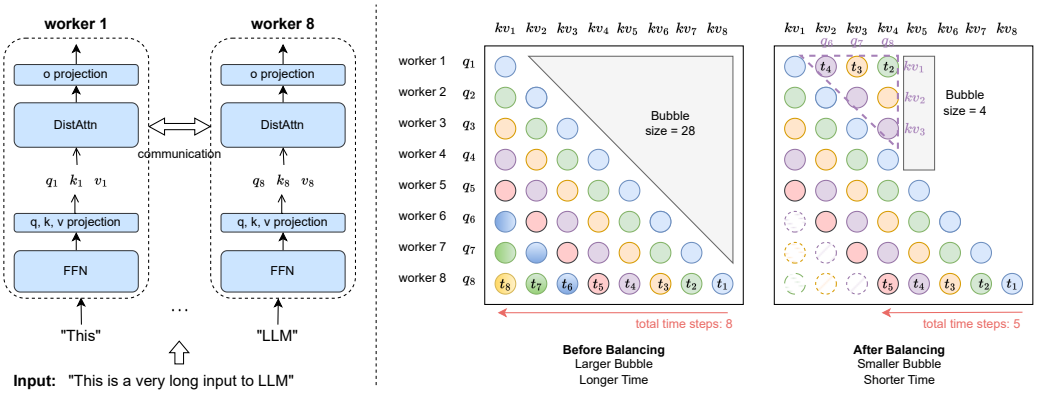

Figure 1: Left: Sequence parallelism in LIGHTSEQ. The input sequence is split into chunks along the sequence dimension and distributed to different workers (8 workers in the illustration). During forward and backward, only the attention module, DISTATTN, requires communication of intermediate tensors like k and v. Some modules like LayerNorm are ignored for simplicity. Right: Illustration of the load-balanced scheduling. "Bubble size" represents the times that a worker is idle. Causal language modeling naturally introduces imbalanced workloads, e.g., worker 1 is idle from time step 2 to time step 8 before balancing. We reduce the bubble fraction by allocating computation from the busy worker (e.g., worker 8) to the idle worker (e.g., worker 1), so worker 1 is only idle at time step 5 after balancing.

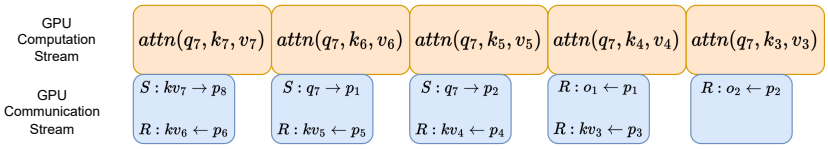

Figure 2: Forward pass example of overlapping communication using worker 7 out of 8 workers. o denotes the attention output computed by a remote worker. For instance, $o_1 = attn(q_7, k_1, v_1)$ for worker 7. In the communication stream, "S" stands for sending, and "R" stands for receiving. For instance, $S : kv_7 \to p_8$ denotes sending the local kv_7 to the remote worker p_8.

inside the computational kernel to avoid materializing large tensors. As a result, many previous recomputation policies become less effective. In this work, we focus on checkpointing at the boundary of every transformer layer, which is a popular strategy adopted by many current open-sourced projects such as FastChat (Zheng et al., 2023).

3 Method

In this section, we describe the design of the key components in LIGHTSEQ. We first introduce a distributed memory-efficient attention, DISTATTN (§3.1) which parallelizes the computation along the sequence dimension. We then introduce a load-balanced scheduling for causal language modeling to reduce the computation bubble as well as an asynchronous communication design that overlaps the communication into computation (§3.2). Finally, we propose a rematerialization-aware checkpointing strategy (§3.3) which effectively cuts off the recomputation time in gradient checkpointing.

3.1 DISTATTN: distributed memory-efficient attention

The core idea in DISTATTN is to split the input sequence consisting of N tokens evenly across P workers (e.g. GPUs) along the sequence dimension. Each worker is therefore responsible for computing the forward and backward pass for only N/P of the N tokens. For modules like the Feed Forward Layer (FFN), Layer Norm (LN), and the embedding layer the tokens can be computed independently without coordination (embarrasingly parallel) and the work is balanced across workers.

Unfortunately, for the attention modules where local tokens may need to attend to remote tokens, coordination is required. To address this, each worker collects all the keys and values associated with other tokens and then locally computes the attention following Dao (2023). To address the memory pressure introduced by collecting all other keys and values, this process is done online by streaming the key and values from workers with earlier tokens to workers with later tokens. More formally, denote $\mathbf{q}_p, \mathbf{k}_p, \mathbf{v}_p$ as the query, key, value inputs held on the p-th worker ($p = \{1, \cdots, P\}$), denote $attn(\mathbf{q}_p, \mathbf{k}_{p'}, \mathbf{v}_{p'})$ as the attention computation w.r.t. p-th chunk of the query and p'-th chunk of the key and value, denote $p_{\text{local}} \in \{1, \cdots, P\}$ as the local rank, and denote $p_{\text{remote}} \in \{1, \cdots, P\}$ as one of the remote ranks. Figure. 1 ("Before Balancing") shows the vanilla version of DISTATTN, where each worker computes the attention for $\mathbf{q}_{p_{\text{local}}}$ and loops over both the local and the remote key and value blocks. We fetch $\mathbf{k}_{p_{\text{remote}}}$ and $\mathbf{v}_{p_{\text{remote}}}$ from rank p_{remote} before the computation of $attn(\mathbf{q}_{p_{\text{local}}}, \mathbf{k}_{p_{\text{remote}}}, \mathbf{v}_{p_{\text{remote}}})$. In Appendix. 6.1, we provide pseudo-code on how to use DISTATTN in LIGHTSEQ, on the p-th worker where there are P total workers.

3.2 Load balanced scheduling with communication and computation overlap

Load balanced scheduling. Causal language modeling objective (Brown et al., 2020; Touvron et al., 2023) is one of the most prevalent objectives for LLMs, where each token only attends to its previous tokens. This naturally introduces a work imbalance between workers in our block-wise attention: as shown in Figure 1 ("Before Balancing"), in an 8-worker ($P = 8$) scenario, the last worker needs to attend to tokens on all other 7 workers, while the first worker is idle after attending to its local tokens, which results in a total idle time of 28. In a general form, the idle fraction is $\frac{P^2-P}{2P^2}$ ($\rightarrow \frac{1}{2}$ when $P \rightarrow \infty$), which means roughly half of the workers are idle. To reduce this idle time (a.k.a., the bubble time), we let early workers that have finished their computation for local $\mathbf{q}_{p_{\text{local}}}$ to help compute for $\mathbf{q}_{p_{\text{remote}}}$ of the later workers. For instance, we let worker 1 compute $attn(\mathbf{q}_8, \mathbf{k}_1, \mathbf{v}_1)$ and send the result to worker 8. When the number of workers is odd, the idle fraction is 0. When the number of workers is even, the idle fraction is $\frac{1}{2P}$, which is asymptotically 0 when scaling to more number of workers.

Communication and computation overlap. DISTATTN relies on peer-to-peer (P2P) communication to fetch the $\mathbf{k}, \mathbf{v}$ (or $\mathbf{q}$ chunks in the load balanced scheduling) from remote devices before computing the corresponding attention block. However, these communications can be easily overlapped with the computation of the former blocks. For instance, When the first worker is computing attention for its local token, it can pre-fetch the next chunk of tokens it needs for the next time step. In modern accelerators, this can be done by placing the attention computation kernel in the main GPU stream, and the P2P communication kernel in another stream, where they can run in parallel (Zhao et al., 2023). We demonstrate the overlapped scheduling for worker 7 on the 8 workers example in Figure. 2. Empirically, we find this optimization greatly reduces the communication overhead (§4.3).

3.3 Rematerialization-aware checkpointing strategy

The de-facto way of training transformers requires gradient checkpointing. Often, the system uses heuristics to insert gradient checkpoints at each Transformer layer (Wolf et al., 2019). However, with the presence of Dao et al. (2022), we found the previous gradient checkpointing strategy will cause an extra recomputation of the flash attention forward kernel. Concretely, when computing the gradient of the MLP layer, Wolf et al. (2019) will re-compute the forward of the entire Transformer layer, including the one in flash attention. However, when computing the gradient of the flash attention kernel, it needs to re-compute the forward of the flash attention again. Essentially, this is because flash attention will not materialize the intermediate values during the forward, and will recompute it during the backward, regardless of the re-computation strategy in the outer system level. To tackle this, we propose to insert checkpoints at the output of the flash attention kernel, instead of at the Transformer

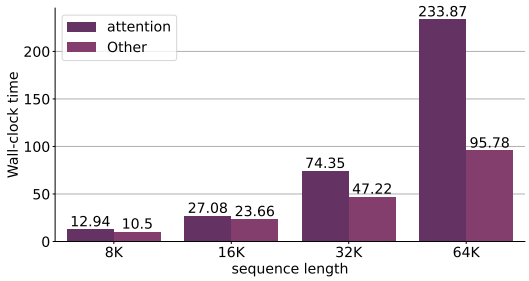

Figure 3: Time breakdown of attention versus other modules in a forward pass. Time measured with Flash-Attention (Dao, 2023) (Unit ms).

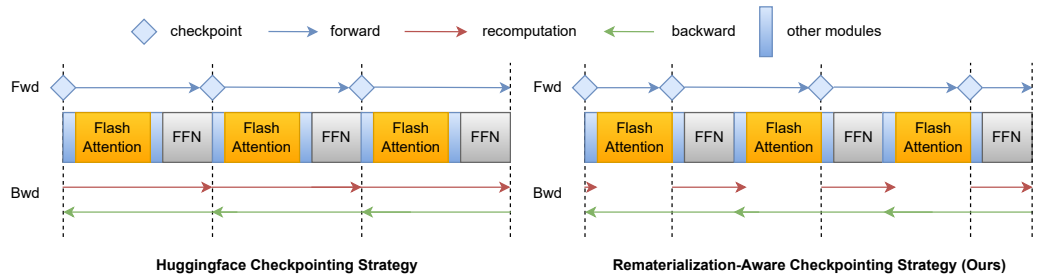

Figure 4: Comparison of HuggingFace gradient checkpointing strategy and our materialization-aware gradient checkpointing strategy. Note that our checkpointing strategy **saves an entire flash attention forward per layer** in recomputation.

layer boundary. In this case, we only need to recompute the forward of flash attention once, effectively saving a forward of attention for each Transformer layer as shown in Figure. 4. In Figure. 3, we show the attention time dominates in the forward pass when scaling up the sequence length, which indicates our method can save $\sim 0.23 \times 32$ (i.e., ~ 7) seconds when training a 64K sequence example on Llama-7b using the local version of flash attention. In addition, this saves a communication brought by our DISTATTN forward in the distributed training scenario. We benchmark the end-to-end speedup brought by this materialization-aware checkpointing strategy in §4.3.

Communication and memory analysis Denote the hidden dimension as d. In DISTATTN, every worker needs to fetch key and value chunks both of size $\frac{N}{P}d$ before performing the corresponding chunk-wise computation. Thus, the total communication volume in the P-workers system is $2 \times \frac{N}{P}d \times P = 2Nd$. With the causal language objective, half of the keys and values do not need to be attended, halving the forward communication volume to Nd. In the backward pass, DISTATTN needs to communicate keys, values, and their gradients, which has $2Nd$ volume. It adds up to $3Nd$ as the total communication volume for DISTATTN. In Megatron-LM (Korthikanti et al., 2023), each worker needs to perform six all-gather and four reduce-scatter on a $\frac{N}{P}d$ size tensor, thus giving a total communication volume of $10Nd$. Considering gradient check-pointing, Megatron-LM will perform communication in the forward again, giving a total volume of $14Nd$. On the other hand, our communication volume remains $3Nd$ because of the rematerialization-aware strategy. In conclusion, LIGHTSEQ achieves 4.7x communication volume reduction compared with Megatron-LM.

In practice, we combine LIGHTSEQ with FSDP to also distribute the model weights for large models. We note that the communication introduced by FSDP is only proportional to the size of model weights, which does not scale up with long sequence length. We show the end-to-end speedup with FSDP in Table 1. In the situations where the model uses MQA or GQA, LIGHTSEQ further saves the communication volumes by the shared key and values, which we discuss in detail in § 4.1. However, we also note that this is a theoretical analysis, where the wall-clock time may differ because of factors such as implementations. In the experiment section, we provide wall-clock end-to-end results for comparison.

4 Experiments

In this section, we evaluate LIGHTSEQ against Megatron-LM (Korthikanti et al., 2023) and show:

1. LIGHTSEQ has faster training speed on a wide range of models. It achieves up to 2.01× speedup over Megatron-LM on various MHA and GQA models.
2. LIGHTSEQ supports longer sequence length by scaling beyond the number of attention heads. We show our method can support 2x-8x longer sequences than Megatron-LM.

In the ablation study, we provide the gain from each component of LIGHTSEQ: Load balancing, computation-communication overlapping, and rematerialization-aware checkpointing.

Cluster setup. We evaluate our method and the baseline in (1) A single A100 DGX box with 8x80 GB GPUs. These GPUs are connected with NVLink; (2) 2 DGX boxes with the same setting. These

two boxes are interconnected by 100 Gbps Infiniband. This is representative of cross-node training, where the communication overhead has a larger effect. (3) Our in-house cluster with 2x8 A100 40GB GPUs without Inifiniband. We report some results on this cluster where conclusions can be drawn from a single-node setup or without involving cross-node training time.

Model setup. We evaluate our system on Llama-7B and its variants of different representative families: (1) Multi-head attention(MHA) models: LLama-7B with 4096 hidden size and 32 query(key and value) heads (Touvron et al., 2023); (2) Grouped-Query attention (GQA) models: Llama-GQA, same as Llama-7B but with 8 key and value heads; (3) models with more general number of attention heads: Llama-33H same as Llama-7B but with 33 query (key and value) attention heads. (4) models with fewer attention heads: we design Llama-16H, Llama-8H, Llama-4H, Llama-2H with 16, 8, 4, and 2 heads. According to Liu et al. (2021), we keep the number of attention heads by scaling the number of layers properly and keep the intermediate FFN layer size the same to make the model sizes still comparable. For example, Llama-16H has 16 attention heads per layer, a hidden size of 2048, an FFN layer of size 11008, and 64 layers.

Implementation. LIGHTSEQ is a lightweight scheduling level prototype. In particular, we implement the load balancing and overlapping in Python and NCCL Pytorch bindings in 1000 lines of codes (Paszke et al., 2019; Jeaugey, 2017), and the checkpointing strategy in 600 lines of Pytorch. It is attention backend agnostic. To reduce the memory consumption and reach faster speed in the attention module, we use the FlashAttention2 algorithm (Dao, 2023). We use the triton (Tillet et al., 2019) implementation and minimally modify it to keep around statistics in the flash attention algorithm. We tweak all block sizes to 128 and the number of stages to 1 for the best performance in our cluster. We reuse the C++ backward kernels of FlashAttention2 because we do not need to modify the backward logic. We run LIGHTSEQ using FSDP to reduce the memory footprint of data parallelism (Zhao et al., 2023). For fair comparisons, we run all comparisons using the same attention backend. We also add support for Megatron-LM so that comparing with them can produce a more insightful analysis: (1) not materializing the causal attention mask, greatly reducing the memory footprint. For instance, without this support, Megatron-LM will run out of memory with Llama-7B at a sequence length of 16K per GPU. (2) head padding where the attention heads cannot be divided by device number. All results are gathered with Adam optimizer, 10 iterations of warm-up, and averaged over the additional 10 iterations.

Table 1: Per iteration wall-clock time of LIGHTSEQ and Megatron-LM (Korthikanti et al., 2023) (Unit: seconds). Speedup in bold denotes the better of the two systems in the same configuration.

Method	# GPUs	Sequence Length		Llama-7B		Llama-GQA		Llama-33H	
		Per GPU	Total	Time	speedup	Time	speedup	Time	speedup
Megatron-LM	1x8	4K	32K	2.54	1.0x	2.43	1.0x	3.15	1.0x
	1x8	8K	64K	6.81	1.0x	6.60	1.0x	8.37	1.0x
	1x8	16K	128K	20.93	1.0x	20.53	1.0x	25.75	1.0x
	1x8	32K	256K	72.75	1.0x	71.93	1.0x	90.21	1.0x
LIGHTSEQ	1x8	4K	32K	2.50	**1.02x**	2.30	**1.06x**	2.58	**1.22x**
	1x8	8K	64K	5.98	**1.14x**	5.61	**1.18x**	6.08	**1.38x**
	1x8	16K	128K	17.26	**1.21x**	16.86	**1.22x**	17.77	**1.45x**
	1x8	32K	256K	58.46	**1.24x**	57.01	**1.26x**	59.96	**1.50x**
Megatron-LM	2x8	4K	64K	5.29	**1.0x**	5.26	1.0x	7.52	1.0x
	2x8	8K	128K	14.26	1.0x	14.21	1.0x	20.63	1.0x
	2x8	16K	256K	43.44	1.0x	43.20	1.0x	62.78	1.0x
	2x8	32K	512K	147.06	1.0x	146.38	1.0x	216.70	1.0x
LIGHTSEQ	2x8	4K	64K	6.85	0.77x	4.92	**1.07x**	7.03	**1.07x**
	2x8	8K	128K	12.75	**1.12x**	9.74	**1.46x**	13.12	**1.57x**
	2x8	16K	256K	30.21	**1.44x**	28.49	**1.52x**	31.33	**2.00x**
	2x8	32K	512K	106.37	**1.38x**	102.34	**1.43x**	107.76	**2.01x**

4.1 faster training speed and better support for different model architectures

In this section, we compare our method with Megatron-LM on three settings: (1) the multi-head attention (MHA) models where the number of key and value heads equals the number of query heads;

Table 2: The maximal sequence length Per GPU supported by LIGHTSEQ and Megatron-LM with tensor parallelism and pipeline parallelism on 16xA100 40GB GPUs. LIGHTSEQ supports 512K sequence length in all models, while Megatron-LM strategy maximal sequence length decreases with fewer heads, with either data parallelism or pipeline parallelism.

	Llama-16H	Llama-8H	Llama-4H	Llama-2H
Megatron TP+DP	512K	256K	128K	64K
Megatron-LM TP+PP	512K	256K	256K	128K
LIGHTSEQ	512K	512K	512K	512K

(2) the grouped-query attention (GQA) models where the number of key and value heads is less than the number of query heads; (3) the models with arbitrary numbers of heads, i.e. the number heads is unnecessarily a multiple of the parallelism degree.

Multi-head attention (MHA). On the Llama-7B model, our method achieves **1.24**× and **1.44**× speedup compared to Megatron-LM in single node and cross node setting, up to the longest sequence length we experiment. This is a joint result of our overlapping communication technique and our rematerialization-aware checkpointing strategy. We analyze how much each factor contributes to this result in the ablation study (§ 4.3). We do note that our method does not achieve better performance in shorter sequences, such as per GPU 4K setting for cross node. This is because the communication dominates the training run-time, where our overlapping technique has not been able to reduce much. We leave the optimization of P2P communication on MHA models and shorter sequence length as an exciting future work.

Grouped-query attention (GQA). On LLama-GQA model, our method achieves better speedup because our communication of key and value vectors significantly reduces. Note that our communication time is proportional to the sum of query, key, value, and output (for load balancing) vectors, where reducing key and value sizes to 8 almost half-en our communication time. On the contrary, the communication time in Megatron-LM does not decrease because its communication happens outside of the attention module, i.e. not influenced by optimization inside the attention module. Thus, its overall training run-time does not decrease as much as LIGHTSEQ.

We take the 4K per-GPU sequence length and 2x8 GPUs as an example for analysis. In the MHA experiment, the communication in a forward and a backward pass of a single attention module is roughly 143ms and the computation time is roughly 53ms. In addition, our overlapping technique is able to hide 45ms into the computation, resulting in a total run-time of 151ms and a net communication overhead of 98 ms. As a reference, the communication in Megatron-LM takes 33ms, which is why Megatron-LM is faster than LIGHTSEQ under this particular setting in the MHA experiment. When considering the GQA case, the communication in LIGHTSEQ roughly reduces to 71 ms. Overlapping with the computation, the communication overhead is now less than that of Megatron-LM. Combined with the checkpointing technique, we are seeing a positive speedup gain at 4K per-GPU sequence length. As the sequence length increases, our overlapping technique, driven by the fact that computation time surpasses communication time, and our checkpointing method, due to the rising ratio of a single attention forward, both contribute to greater speedup. Overall, we can observe speedups up to **1.52**× on the cross-node setting, making an additional eight percent enhancement compared to the results in the MHA experiment of the same setting.

In support of arbitrary numbers of heads. With Llama-33H models, Megatron-LM exhibits an additional performance decline compared to LIGHTSEQ. This is due to its requirement to pad the number of attention heads so that the number of attention heads is divisible by the number of devices. On the other hand, LIGHTSEQ does not need to partition attention heads and can support an arbitrary number of heads efficiently. For instance, when using 8 GPUs, Megatron-LM must pad the attention heads to 40, resulting in 21.2% of the computation being wasted. In the case of 16 GPUs, Megatron-LM is compelled to pad the attention heads to 48, leading to a more substantial computation wastage of 45.5%. This roughly corresponds to a 1.21× or 1.45× increase in run-time compared to LIGHTSEQ when training a Llama-7B model. This performance degradation of Megatron-LM is primarily because the training time is dominated by the attention module's computation time when

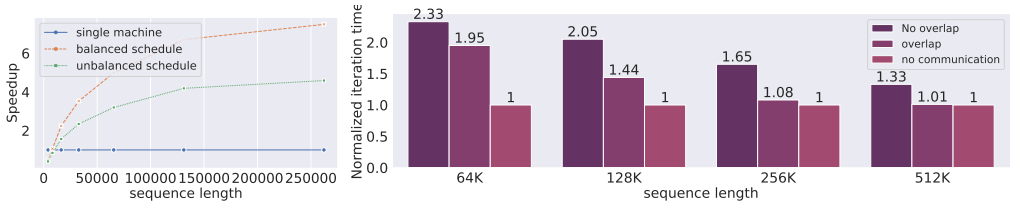

Figure 5: Ablation on the effect of balanced schedule (left) and the effect of overlapping (right).

scaling to longer sequence lengths. Empirically, we observe a **1.50**× and **2.01**× speedup (an additional 20% and 45% speedup compared to Llama-7B cases, aligned with the theoretical analysis).

4.2 Scaling beyond the number of heads.

Assuming the number of heads being a multiple of the tensor parallelism degree constraints Megatron-LM to scale its tensor parallelism degree beyond the number of heads, thus limiting its scaling ability to longer sequence lengths. When the number of GPUs exceeds the number of attention heads, there will be three possible solutions to use Megatron-LM. First, the user can pad dummy heads as in the Llama-33H scenario. However, when scaling to longer sequences, the percentage of dummy heads padded almost directly translates to the percentage of slowdown. For instance, for Llama-8H, this solution pads 2× dummy heads and would almost translate to a 2× slowdown, which is very inefficient. Second, the user can use data parallelism for excess GPUs. For instance, a user with 16 GPUs can choose to use 4-way data parallelism and 4-way tensor parallelism on the Llama-4H model. Since data parallelism does not partition the activation, the system can only support sequences as if the user only has 4 GPUs. Lastly, the user may choose to use pipeline parallelism to partition activation. However, the memory usage at each stage of the pipeline is not evenly distributed, still limiting the maximal sequence length supported. In particular, the first pipeline stage usually stores more activations because it will hold the most active micro-batches. For instance, in the Llama-2H experiment, we find that different stages consume from 18GB to 32GB in a 64K sequence length. In addition, using pipeline parallelism introduces an extra fraction of GPU idle time. We demonstrate the effect of using the latter two solutions in Table 2. In 16 A100 40GB GPUs, LIGHTSEQ supports the training of 2× and 8× longer sequences.

4.3 Ablation Study

Effect of load balancing. We study the effect of load balancing using the forward pass of an attention operation in Llama-7B model, on 8 A100 40GB GPUs. The backward pass follows a similar analysis. With an unbalanced schedule (Figure 1), the total work done is 36, where the total work could be done in 8 units of time is 64. Thus, the expected maximal speedup is 4.5x. In the balanced schedule, the expected maximal speedup is 7.2x. We scale the total sequence length from 4K to 256K. The unbalanced version saturates in 4.5x speedup compared to a single GPU implementation, while the balanced version saturates 7.5x [2] speedup. Both of them align with our earlier theoretical analysis and show the importance of our balanced scheduling.

Effect of overlapping communication and computation. We study the benefits of overlapping communication on Llama-7B and 2 DGX boxes. We find that overlapping greatly reduce the communication overhead. For instance, on a global sequence length of 128K, the communication overhead is reduced from 105% to 44%. This overlapping scheme maximizes its functionality when the communication overhead is less than 100%, where all communication can be potentially overlapped. Empirically, we find the system only exhibits 8% and 1% overhead in these cases, showing a close performance to an ideal system without communication.

Effect of materialization-aware checkpointing. We show in Table. 3 the ablation results of our rematerialization-aware gradient checkpointing. Our method achieves 1.16x, 1.24x, and 1.31x

[2]We find the single machine attention flops drop with very long sequence length, resulting in a slightly higher speedup than assuming its perfect scalability.

speedup at the sequence length of 8K, 16K, and 32K per GPU respectively. The materialization-aware checkpointing strategy speeds up more at longer sequence lengths because it saves an entire attention forward which dominates the computation at longer sequence lengths.

Table 3: Ablation study on the effect of the rematerialization-aware gradient checkpointing on 8 A100s in a single node with a batch size of 1. We report the end-to-end run time in seconds and show the speedup of our gradient checkpointing strategy ("Our ckpt") over the HuggingFace gradient checkpointing strategy ("HF ckpt").

Ckpt Method	Sequence Length Per GPU					
	1K	2K	4K	8K	16K	32K
HF ckpt	0.84	1.29	2.64	6.93	21.44	76.38
Our ckpt	0.84	1.36	2.50	5.98	17.26	58.46
Speedup	1.0x	0.94x	**1.06x**	**1.16x**	**1.24x**	**1.31x**

4.4 Discussion

In this section, we first discuss the future directions that can further improve LIGHTSEQ. We then compare our method with one concurrent open-sourced project which also splits the attention heads. Finally, we discuss the role of pipeline parallelism in supporting long sequence training and shows it is less effective than tensor parallelism, which is the reason we do not consider it as a major baseline.

Optimizing P2P communication and better support for shorter context length. As shown in §4.1, LIGHTSEQ may be slower in shorter context length and MHA models (Llama-7B on per GPU sequence length 4K). Based on our preliminary investigation, this is because our usage of P2P is not as optimized as primitives used in tensor model parallelism, such as all-gather kernels. For instance, they are not aware of the underlying cluster topology. In the future, we plan to implement the P2P scheduling in a topology-aware way to further improve the communication time.

Comparison to DeepSpeed Ulysses. DeepSpeed-Ulysses [3] is a concurrent open-sourced implementation, which uses all-to-all communication primitive to reduce the communication volume. In our testing, we verified that their communication is lower than Megatron-LM. Yet, as it is also partitioning the attention head dimension, it suffers from similar problems as analyzed above. We provide some end-to-end comparisons in Appendix 6.2. We note that the communication in DeepSpeed Ulysses can be faster than LIGHTSEQ, especially with shorter context length and slower network, where the overlapping technique in LIGHTSEQ cannot perfectly hide all the communication. This can be potentially addressed by optimizing the P2P communication as discussed above.

Pipeline parallelism. Pipeline parallelism also partitions the activation. However, as mentioned in § 4.2, it does not partition the activations evenly across stage, leaving high memory pressure to the first stage. Thus, we mainly focus on comparing with tensor model parallelism (combined with sequence parallelism) in this work and only consider including pipeline parallelism for comparison when the tensor parallelism is limited by the number of heads.

5 Conclusion

In this work, we introduce LIGHTSEQ, a sequence parallel prototype for long-context transformer training. LIGHTSEQ presents novel system optimizations including load balancing for causal language modelings, overlapped communication with computation in the distributed attention computation, and a re-materialization-aware checkpointing strategy. Our experiments evaluate multiple families of transformer models and on different cluster types, showing that it achieves up to 2.01× speedup and scales up to 8x longer sequences, compared to another popular system, Megatron-LM,. Future directions include implementing topology-aware P2P operations to further reduce training time in lower sequence lengths.

[3] `https://github.com/microsoft/DeepSpeed/tree/master/blogs/deepspeed-ulysses`

References

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Yuchen Bian, Jiaji Huang, Xingyu Cai, Jiahong Yuan, and Kenneth Church. On attention redundancy: A comprehensive study. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 930–945, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

Sylvain Jeaugey. Nccl 2.0. In *GPU Technology Conference (GTC)*, volume 2, 2017.

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

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, and Daniel Haziza. xformers: A modular and hackable transformer modelling library. `https://github.com/facebookresearch/xformers`, 2022.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can open-source llms truly promise on context length, 2023.

Shenggui Li, Fuzhao Xue, Yongbin Li, and Yang You. Sequence parallelism: Making 4d parallelism possible. *arXiv preprint arXiv:2105.13120*, 2021.

Hao Liu and Pieter Abbeel. Blockwise parallel transformer for long context large models. *arXiv preprint arXiv:2305.19370*, 2023.

Liyuan Liu, Jialu Liu, and Jiawei Han. Multi-head or single-head? an empirical comparison for transformer training. *arXiv preprint arXiv:2106.09650*, 2021.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

Anton Osika. gpt-engineer, 2023. URL `https://github.com/AntonOsika/gpt-engineer`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

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

Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. A length-extrapolatable transformer. *arXiv preprint arXiv:2212.10554*, 2022.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

6 Supplementary Material

6.1 Using DISTATTN in LIGHTSEQ

Algorithm 1 DISTATTN in LIGHTSEQ (forward pass)

Require: Matrices $\mathbf{Q}^p, \mathbf{K}^p, \mathbf{V}^p \in \mathbb{R}^{\frac{N}{\mathbb{P}} \times d}$ in HBM, block sizes B_c, B_r, rank

0: **function** STANDALONE_FWD(q, k, v, o, ℓ, m, causal, last)
1: Divide q into $T_r = \left\lceil \frac{N}{\mathbb{P}B_r} \right\rceil$ blocks $_1, \ldots, _{T_r}$ of size $B_r \times d$ each, and divide k, v in to $T_c = \left\lceil \frac{N}{\mathbb{P}B_c} \right\rceil$ blocks $k_1, \ldots, k_{T_c}$ and $v_1, \ldots, v_{T_c}$, of size $B_c \times d$ each.
2: Divide the output $o \in \mathbb{R}^{\frac{N}{\mathbb{P}} \times d}$ into T_r blocks $o_i, \ldots, o_{T_r}$ of size $B_r \times d$ each, and divide the logsumexp L into T_r blocks $L_i, \ldots, L_{T_r}$ of size B_r each.
3: **for** $1 \le i \le T_r$ **do**
4: Load q_i from HBM to on-chip SRAM.
5: Load $o_i \in \mathbb{R}^{B_r \times d}$, $\ell_i \in \mathbb{R}^{B_r}$, $m_i \in \mathbb{R}^{B_r}$ from HBM to on-chip SRAM as $o_i^{(0)}$, $\ell_i^{(0)}$, $m_i^{(0)}$.
6: **for** $1 \le j \le T_c$ **do**
7: **if** causal and $i \le j$ **then**
8: Continue
9: **end if**
10: Load k_j, v_j from HBM to on-chip SRAM.
11: On chip, compute $s_i^{(j)} = q_i k_j^T \in \mathbb{R}^{B_r \times B_c}$.
12: On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \text{rowmax}(s_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{p}_i^{(j)} = \exp(S_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \text{rowsum}(\tilde{p}_i^{(j)}) \in \mathbb{R}^{B_r}$.
13: On chip, compute $o_i^{(j)} = \text{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} o_i^{(j-1)} + \tilde{p}_i^{(j)} v_j^p$.
14: **end for**
15: On chip, compute $o_i = \text{diag}(\ell_i^{(T_c)})^{-1} o_i^{(T_c)}$.
16: Write o_i to HBM as the i-th block of o.
17: **if** last **then**
18: On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.
19: Write L_i to HBM as the i-th block of L.
20: **end if**
21: **end for**
22: Return o, ℓ, m and the logsumexp L.
22: **end function**
22: Initialize $\mathbf{O}^p = (0)_{\frac{N}{\mathbb{P}} \times d} \in \mathbb{R}^{\frac{N}{\mathbb{P}} \times d}$, $\ell^{(p)} = (0)_{\frac{N}{\mathbb{P}}} \in \mathbb{R}^{\frac{N}{\mathbb{P}}}$, $m^p = (-\infty)_{\frac{N}{\mathbb{P}}} \in \mathbb{R}^{\frac{N}{\mathbb{P}}}$.
22: $\mathbf{O}^p, \ell^p, m^p, L^p$ = standalone_fwd($\mathbf{Q}^p, \mathbf{K}^p, \mathbf{V}^p, \mathbf{O}^p, \ell^p, m^p$, True, p=1)
23: **for** $1 \le r < p$ **do**
24: Receive $\mathbf{K}^r$ and $\mathbf{V}^r$ from **Remote** worker r into HBM.
25: $\mathbf{O}^p, \ell^p, m^p, L^p$ = standalone_fwd($\mathbf{Q}^p, \mathbf{K}^y, \mathbf{V}^y, \mathbf{O}^p, \ell^p, m^p$, False, r=(p-1)
26: Delete $\mathbf{K}^r$ and $\mathbf{V}^r$ from HBM.
27: **end for**
28: Return the output $\mathbf{O}^p$ and the logsumexp L. =0

In this section, we provide more details of DISTATTN, and how it can be used with the outer LIGHTSEQ logic of the forward pass (Alg 1). For conceptual simplicity, we demonstrate it in the most vanilla version, without the actual scheduling (e.g. load balancing and overlapping). We also demonstrate it with the causal language modeling objective. The standalone attention is mainly borrowed from the FlashAttention2 paper (Dao, 2023). To make it compatible with DISTATTN, we mainly revised the several points:

1. Accumulate results statistics o, m and l from previous computation, instead of initializing them inside the function.
2. Pass an extra argument "last", which means whether this is the last chunk of attention computation. Only when it is true, we compute the logsumexp L.

At a high level, on a worker p, LIGHTSEQ first initializes local statistics m, l, L. Then LIGHTSEQ loops over all its previous workers. In each iteration, it fetches the key and the value from a worker and invokes the revised standalone attention to update local statistics. At the end of the iteration, it needs to delete the remote key and value from HBM so that the memory does not accumulate. At the last iteration of the loop, it additionally calculates the logsumexp according to the final m and l (the "last" variable in the algorithm). At the end of the forward pass, worker p has the correct m, l, L. The backward pass is similar and conceptually simpler because we do not need to keep track of statistics such as m and l. Instead, we only need to use the logsumexp stored in the forward pass.

6.2 Comparison with DeepSpeed Ulysses

Method	# GPUs	Sequence Length		Time	Speedup
		Per GPU	Total		
Llama-7B					
Megatron-LM	2x8	4K	64K	5.29	1.0x
	2x8	8K	128K	14.26	1.0x
	2x8	16K	256K	43.44	1.0x
	2x8	32K	512K	147.06	1.0x
DeepSpeed-Ulysses	2x8	4K	64K	4.29	**1.23x**
	2x8	8K	128K	11.61	**1.23x**
	2x8	16K	256K	37.53	1.16x
	2x8	32K	512K	134.09	1.10x
LIGHTSEQ	2x8	4K	64K	6.85	0.77x
	2x8	8K	128K	12.75	1.12x
	2x8	16K	256K	30.21	**1.44x**
	2x8	32K	512K	106.37	**1.38x**
Llama-33H					
Megatron-LM	2x8	4K	64K	7.52	1.0x
	2x8	8K	128K	20.63	1.0x
	2x8	16K	256K	62.78	1.0x
	2x8	32K	512K	216.70	1.0x
DeepSpeed-Ulysses	2x8	4K	64K	6.42	**1.17x**
	2x8	8K	128K	17.47	1.18x
	2x8	16K	256K	56.63	1.11x
	2x8	32K	512K	202.89	1.07x
LIGHTSEQ	2x8	4K	64K	7.03	1.07x
	2x8	8K	128K	13.12	**1.57x**
	2x8	16K	256K	31.33	**2.00x**
	2x8	32K	512K	107.76	**2.01x**

Table 4: Per iteration wall-clock time of LIGHTSEQ, Megatron-LM (Korthikanti et al., 2023) and DeepSpeed Ulysses (Unit: seconds). Speedup in bold denotes the better of the three systems. We calculate the speedup based on Megatron-LM iteration time.

We run a subset of the experiments compared with DeepSpeed-Ulysses. Firstly, DeepSpeed-Ulysses does reduce the communication overhead, and thus better than Megatron-LM on scenarios listed in Table 4. LIGHTSEQ achieves better performance than DeepSpeed-Ulysses on longer sequences or models with a more general number of heads (e.g. Llama-33H). We also note that DeepSpeed-Ulysses can not scale beyond the number of attention heads because it also relies on sharding the attention heads. However, we need to point out that in shorter sequences and MHA models (where LIGHTSEQ does not have a communication advantage, compared to GQA/MQA models), the communication primitives used in DeepSpeed-Ulysses are more advantageous. We leave our further optimization in P2P in shorter sequences and MHA models as an exciting future work.