OPT2024: 16th Annual Workshop on Optimization for Machine Learning

Linear Attention Sequence Parallelism

Weigao Sun* SUNWEIGAO@OUTLOOK.COM
Equal Contribution, Shanghai AI Laboratory

Zhen Qin* ZHENQIN950102@GMAIL.COM
Equal Contribution, TapTap

Dong Li LIDONG@PJLAB.ORG.CN
Shanghai AI Laboratory

Xuyang Shen SHENXUYANG@PJLAB.ORG.CN
Shanghai AI Laboratory

Yu Qiao QIAOYU@PJLAB.ORG.CN
Shanghai AI Laboratory

Yiran Zhong† ZHONGYIRAN@GMAIL.COM
Corresponding Author, Shanghai AI Laboratory

Abstract

Sequence parallel serves as a prevalent strategy to handle long sequences that exceed the memory limit of a single GPU. However, existing methods do not take advantage of linear attention features, resulting in sub-optimal parallelism efficiency and usability for linear-complexity language models. In this paper, we introduce Linear Attention Sequence Parallel (LASP), an efficient sequence parallel method designed for linear attention-based language models. Specifically, we design an efficient point-to-point communication mechanism to leverage the right-product kernel trick of linear attention, which sharply decreases the communication overhead. We enhance the practical efficiency of LASP by performing kernel fusion and intermediate state caching, making the implementation of LASP hardware-friendly on GPU clusters. Furthermore, we meticulously ensure the compatibility of sequence-level LASP with all types of batch-level data parallel methods, which is vital for distributed training on large clusters with long sequences and large batches. We also discuss the versatility of LASP on other linear-complexity models. Extensive experiments on linear attention-based models are conducted with varying sequence lengths and GPU cluster sizes. LASP scales sequence length up to 4096K using 128x A100 80G GPUs on 1B models, which is 8× longer than existing methods while being significantly faster.

1. Introduction

Recently, linear-complexity sequence modeling methods [1, 12, 13] are becoming increasingly popular due to their faster processing speed and comparable modeling performance to vanilla Softmax transformers [18–21]. As the size of large language models (LLMs) increases and sequence lengths extend, the capacity limitations of single GPU's memory become a significant challenge, constraining the maximum sequence length manageable by a language model. To address this, Sequence Parallel (SP) techniques [8, 11] are employed, which partition a long sequence into multiple sub-sequences to be processed on separate GPUs. However, current implementations of SP methods do not fully

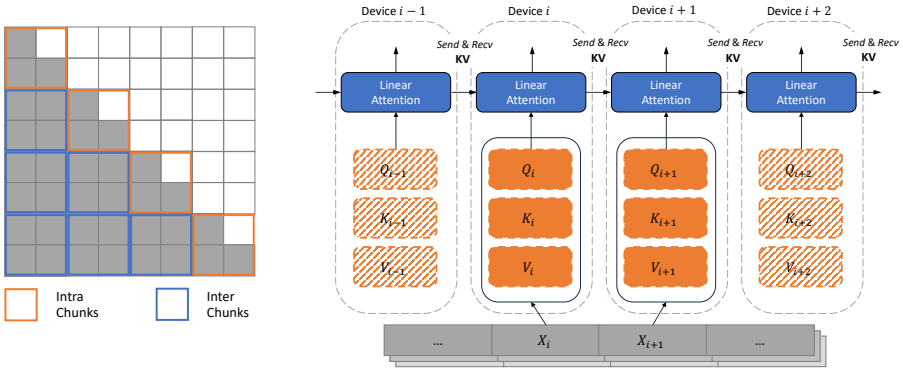

Figure 1: **Visualization of LASP.** This figure illustrates the P2P communication mechanism employed by LASP. The complete input sequence $\mathbf{X}$ is divided into multiple sub-sequence chunks $\{\cdots, \mathbf{X}_i, \mathbf{X}_{i+1}, \cdots\}$, each processed by different model instances across distinct devices. For each device i, Q_i, K_i, and V_i are computed from its respective input chunk $\mathbf{X}_i$. This setup facilitates the execution of linear attention calculations. Notably, the communication operations between devices are designed to be complementary in the forward and backward passes. Specifically, in the forward pass, $\mathbf{KV}$ matrices are sent from device i to device $(i+1)$, and in the backward pass, $\mathbf{dKV}$ matrices are sent back from device $(i+1)$ to device i.

exploit the advantages of linear-complexity attention mechanisms. This results in less than optimal parallelism efficiency and reduced usability.

In this paper, we present the Linear Attention Sequence Parallel (LASP) technique for efficient sequence parallelism on models with linear-complexity "Attention". Our approach takes linear attention as an instance to design a sophisticated communication mechanism based on point-to-point (P2P) communication for exchanging intermediate states during forward and backward passes among GPUs within a node or across multiple nodes. This design maximizes the utilization of right-product kernel tricks [5] in linear attention. Notably, our technique is independent of attention heads partitioning, which allows it to be applied to models with varying numbers or styles of attention heads, such as multi-head, multi-query, and grouped-query attentions. This flexibility exceeds the capabilities of existing SP methods in Megatron-LM [8, 15] or DeepSpeed [4].

Our implementation of LASP incorporates system engineering optimizations such as kernel fusion and KV State caching, resulting in significantly enhanced execution efficiency. Furthermore, we have taken great care in ensuring compatibility of LASP with various (sharded) distributed data-parallel (DDP) [10] training methods during the implementation, which we refer to as the data-sequence hybrid parallelism. Through extensive experiments with linear transformer models of different parameter numbers, cluster sizes, and sequence lengths, we demonstrate LASP's performance and efficiency when used with these DDP instances. Specifically, LASP is significantly faster than existing SP methods and can extend sequence length $8\times$ longer under the same hardware constraints.

Our primary contributions can be summarized as follows:

- *A new SP strategy called LASP that is designed for linear-complexity sequence modeling methods.* LASP is able to perform sequence-level distributed training on $8\times$ longer sequence than existing SP methods while being significantly faster.
- *Sequence length-independent communication overhead.* Our proposed P2P communication mechanism leverages right-product kernel trick of linear attention to ensure that the exchanging of linear attention intermediate states is sequence length-independent.

- *GPU friendly implementation.* We optimize LASP's execution on GPU hardware through meticulous system engineering, including kernel fusion and KV State caching.
- *Data-parallel compatibility.* LASP is compatible with all batch-level DDP methods, including PyTorch/Legacy DDP, FSDP, and ZeRO-series optimizers.

2. Method

LASP tiles sequence over the cluster. Follow the thought-of-tiling, LASP partitions the input sequences into multiple sub-sequence chunks, distributing these chunks individually across different GPUs. For linear attention in a casual setting, in order to fully exploit the advantage of right-product in linear attention, we categorize the attention computation for chunks into two distinct types: intra-chunks and inter-chunks. Intra-chunks involve conventional attention computation, while inter-chunks leverage the kernel tricks associated with linear attention's right-product. Further details regarding the intricate mechanisms of LASP in data distribution, forward pass, and backward pass are expounded upon below. A visualization of LASP is presented in Fig. 1.

Algorithm 1: LASP (Forward Pass)

Input: input sequence in embedding space $\mathbf{X} \in \mathbb{R}^{N \times d}$ with sequence length N and hidden dimension d, distributed world size W, sequence parallel size $T = W$, decay rate $\lambda \in \mathbb{R}^{+}$;
Distribute input sequence $\mathbf{X}$ according to Algorithm 2;
Obtain sub-sequence length (or chunk size) $C = N/T$;
Initialize mask $\mathbf{M} \in \mathbb{R}^{C \times C}$, where $M_{ij} = \lambda^{i-j}$, if $i \geq j$, else $M_{ij} = 0$;
Initialize $\mathbf{\Lambda} = \text{diag}\{\lambda, \lambda^2, \cdots, \lambda^C\} \in \mathbb{R}^{C \times C}$;
Initialize activation state $\mathbf{KV} = \mathbf{0} \in \mathbb{R}^{d \times d}$;
for *chunk* $t \in \{1, \cdots, T\}$ *at rank* $i \in \{1, \cdots, W\}$ *in parallel* **do**
  Calculate $\mathbf{Q}_t = \mathbf{X}_t\mathbf{W}_Q$, $\mathbf{K}_t = \mathbf{X}_t\mathbf{W}_K$, $\mathbf{V}_t = \mathbf{X}_t\mathbf{W}_V$ according to its own data chunk, of size $C \times d$ for each;
  Compute $\mathbf{O}_{\text{t,intra}} = [(\mathbf{Q}_t\mathbf{K}_t^\top) \odot \mathbf{M}]\mathbf{V}_t$;
end
for *chunk* $t \in \{1, \cdots, T\}$ *at rank* $i \in \{1, \cdots, W\}$ **do**
  `Recv` activation $\mathbf{KV}_{t-1}$ from rank $(i-1)$;
  Save $\mathbf{KV}_{t-1}$ as $\mathbf{KV}_i$ on rank i for backward computation;
  Compute $\mathbf{O}_{\text{t,inter}} = \mathbf{\Lambda}\mathbf{Q}_t\mathbf{KV}_{t-1}$;
  Compute $\mathbf{O}_t = \mathbf{O}_{\text{t,intra}} + \mathbf{O}_{\text{t,inter}}$ as $\mathbf{O}$ of t-th chunk;
  Update $\mathbf{KV}_t = \lambda^C\mathbf{KV}_{t-1} + (\lambda^C\mathbf{\Lambda}^{-1}\mathbf{K}_t)^\top\mathbf{V}_t$;
  `Send` activation $\mathbf{KV}_t$ to rank $(i+1)$;
end
Return: $\mathbf{O} = [\mathbf{O}_t]$, with $t \in \{1, \cdots, T\}$.

To streamline derivations, the $\text{Norm}(\cdot)$ operator in Eq. (10) is temporarily omitted. Additionally, we consider a normal case where $W = T$, indicating $G = W/T = 1$. In this scenario, GPU with rank 0 consolidates all split sub-sequences in a batch, subsequently distributing them to all GPUs across the entire distributed world. It is noteworthy that the scenario where the sequence parallel size is not equal to world size is discussed in Sec.A.4.

Without loss of generality, we add λ as the decay rate in linear attention with casual mask, choosing $\lambda = 1$ yields the ordinary linear attention [13, 16]. In the forward pass of linear attention computation with casual mask, the s-th output can be formulated as

$$\mathbf{o}_s^\top = \mathbf{q}_s^\top \sum_{i \leq s} \lambda^{s-i}\mathbf{k}_i\mathbf{v}_i^\top. \tag{1}$$

Rewrite in a recurrence form [6], we have

$$\mathbf{kv}_0 = 0 \in \mathbb{R}^{d \times d}, \quad \mathbf{kv}_s = \lambda\mathbf{kv}_{s-1} + \mathbf{k}_s\mathbf{v}_s^\top, \quad \mathbf{o}_s^\top = \mathbf{q}_s^\top(\mathbf{kv}_s), \tag{2}$$

where

$$\mathbf{kv}_s = \sum_{i \leq s} \lambda^{s-i} \mathbf{k}_i \mathbf{v}_i^\top \tag{3}$$

is the activation state in the forward computation of linear attention with s-th input.

In the sequence parallelism scenario, given data chunk $\mathbf{X}_t$ on rank i, the query, key and value corresponding to $\mathbf{X}_t$ is $\mathbf{Q}_t = \mathbf{X}_t \mathbf{W}_Q$, $\mathbf{K}_t = \mathbf{X}_t \mathbf{W}_K$, $\mathbf{V}_t = \mathbf{X}_t \mathbf{W}_V$. Note that we assume $T = W$ here, their indices are thus equivalent, *i.e.,* $t = i$. The output within the t-th chunk can be calculated as

$$\mathbf{O}_{\text{t,intra}} = [(\mathbf{Q}_t \mathbf{K}_t^\top) \odot \mathbf{M}] \mathbf{V}_t. \tag{4}$$

The intra-chunk computation has no dependencies with other chunks on other GPUs, so it can be calculated parallelized on all ranks in the distributed world. However, this result does not consider the impact of the previous $1 \sim (t-1)$ chunks on the t-th chunk, which is called an inter-chunk. To calculate inter-chunk, let us rearrange Eq. (1) as

$$\mathbf{o}_{s+C}^\top = \mathbf{q}_{s+C}^\top \sum_{i \leq s+C} \lambda^{s+C-i} \mathbf{k}_i \mathbf{v}_i^\top = \mathbf{q}_{s+C}^\top \sum_{i=C+1}^{C+s} \lambda^{s+C-i} \mathbf{k}_i \mathbf{v}_i^\top + \lambda^s \mathbf{q}_{s+C}^\top \sum_{i \leq C} \lambda^{C-i} \mathbf{k}_i \mathbf{v}_i^\top. \tag{5}$$

The first part (before the plus sign) in Eq. (5) corresponds to the computation on intra-chunk, and the second part (after the plus sign) corresponds to the computation on inter-chunk. In sequence parallelism scenario, Eq. (5) can be rewritten in the chunk form as follows:

$$\mathbf{O}_{\text{t,inter}} = \mathbf{\Lambda} \mathbf{Q}_t \mathbf{K}\mathbf{V}_{t-1}, \tag{6}$$

where $\mathbf{KV}_t = \mathbf{kv}_{tC}$. It is worth noting that the calculation of the inter part for the t-th chunk depends on the activation state of previous $(t-1)$ chunk, *i.e.,* $\mathbf{KV}_{t-1}$, which is calculated on rank $(i-1)$. Thus a P2P communication operation `Recv` should be performed to pull $\mathbf{KV}_{t-1}$ from rank $(i-1)$ to rank i. Then the activation state $\mathbf{KV}_t$ should be updated for subsequent inter-chunk attention computation at $(t+1)$-th chunk. The update rule of $\mathbf{KV}_t$ at t-th chunk is

$$\begin{aligned} \mathbf{KV}_t &= \sum_{s \leq tC} \lambda^{tC-s} \mathbf{k}_s \mathbf{v}_s^\top = \lambda^C \sum_{s \leq (t-1)C} \lambda^{(t-1)C-s} \mathbf{k}_s \mathbf{v}_s^\top + \sum_{s=(t-1)C+1}^{tC} \lambda^{tC-s} \mathbf{k}_s \mathbf{v}_s^\top \\ &= \lambda^C \mathbf{KV}_{t-1} + \left(\text{diag}\{\lambda^{C-1}, \ldots, 1\} \mathbf{K}_t\right)^\top \mathbf{V}_t = \lambda^C \mathbf{KV}_{t-1} + \left(\lambda^C \mathbf{\Lambda}^{-1} \mathbf{K}_t\right)^\top \mathbf{V}_t. \end{aligned} \tag{7}$$

In correspondence to the preceding `Recv` operation, another P2P communication operation `Send` is executed to transmit the acquired $\mathbf{KV}_t$ in Eq. (7) to the subsequent rank $(i+1)$ for its inter-chunk computation.

It is noteworthy that in the backward pass, the t-th chunk necessitates $\mathbf{KV}_{t-1}$ as activation to calculate gradients. To minimize communication operations, we cache $\mathbf{KV}_{t-1}$ on High-Bandwidth Memory (HBM) to accelerate computation. Integrating both the intra and inter parts, the final forward output is as follows:

$$\mathbf{O}_t = \mathbf{O}_{t,\text{intra}} + \mathbf{O}_{t,\text{inter}} \tag{8}$$

We present the complete expression of forward pass for LASP with $W = T$ in Algorithm 1.

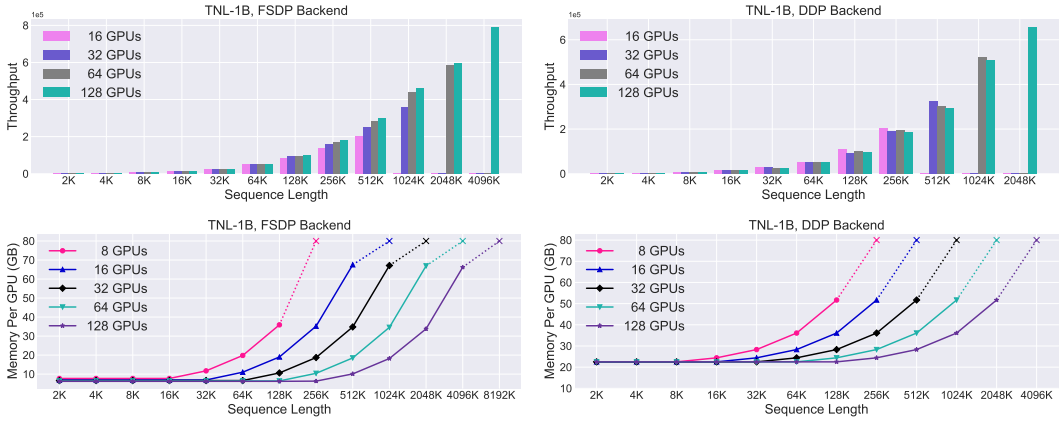

Figure 2: **Scalability Evaluation of LASP on Throughput (tokens/sec) and Memory Usage**. Left: Integration of LASP with FSDP backend; Right: Integration of LASP with DDP backend. The TNL-1B model is used, with a batch size of 1 across up to 128x A100 80GB GPUs. The sign "×" with a dotted line represents occurring an Out of Memory (OOM).

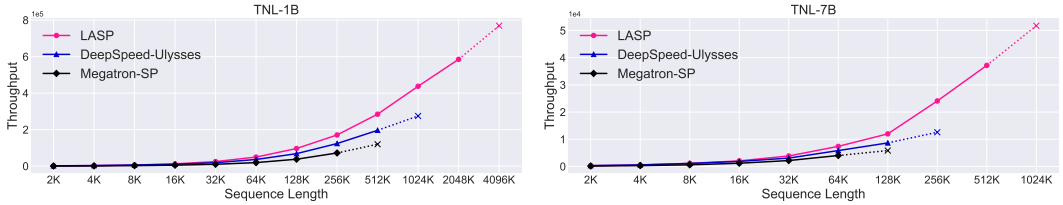

Figure 3: **Speed Comparison (tokens/sec) of LASP Against DeepSpeed-Ulysses and Megatron-SP.** The sign "×" with a dotted line represents occurring an Out of Memory (OOM). The evaluation utilizes the TNL-1B and 7B models with a batch size of 1 on 64x A100 80GB GPUs. The parallelism size for these three methods is configured to 64.

3. Experiments

We evaluate LASP on two representative linear attention-based models: TransNormerLLM (TNL) [13] and Linear Transformer [5]. Our assessment focuses on three key areas: 1) the ability of LASP to scale up sequence length on scaling-out GPUs, 2) speed evaluation when using LASP and its comparison with other SP methods, and 3) the convergence when using LASP. All experiments are conducted on a GPU cluster equipped with 128x A100 80G GPUs. Our implementation is built on Metaseq [22], a PyTorch-based sequence modeling framework with FairScale [2] integrated.

3.1. Scalability and Speed Comparison

The scalability results regarding throughput and memory usage with varying sequence lengths and number of GPUs are illustrated in Fig. 2. By using LASP, we successfully scale the sequence length up to 4096K using the FSDP backend and 2048K with the DDP backend on a TNL model with 1B parameters, on 128 GPUs. We keep using a fixed batch size of 1 to thoroughly assess the performance of LASP across a range of sequence lengths, from a typical 2K to an exceptionally long 4096K. By keeping the batch size constant at 1, we ensure that the experiment results are directly comparable, with only the sequence length varying.

We furthermore conducted a comparison of sequence parallelism on TNL 1B and 7B models against two existing SP methods: DeepSpeed-Ulysses [4] and Megatron-SP [8]. All results presented in Fig. 3 are obtained on 64 GPUs. LASP demonstrates a notable enhancement in throughput for linear attention, primarily due to its efficient communication design that facilitates the exchange of

linear attention intermediate states. Specifically, LASP outperforms DeepSpeed-Ulysses by 38% and Megatron by 136% in terms of throughput at 256K sequence length on 1B model, with the performance gap widening as the sequence length increases. Additionally, system optimizations like kernel fusion and KV State caching enable LASP to support the longest sequence lengths within the same cluster, achieving 2048K for the 1B model and 512K for the 7B model.

4. Conclusion

We presented LASP, effectively addressing the limitations of existing SP methods on linear-complexity models by leveraging the right-product features of linear attention, which significantly enhanced parallelism efficiency and usability. Through the design of an efficient P2P communication mechanism and engineering optimizations including kernel fusion and KV state caching, LASP achieved a notable reduction in communication traffic and improved hardware utilization on GPU clusters. Compatibility with all types of batch-level DDP methods ensured the practicability of LASP for large-scale distributed training. Our experiments highlighted the advantages of LASP on scalability, speed, memory usage and convergence performance. In specific experimental setup, LASP achieves 38% and 136% faster sequence-level distributed training speed on a maximum 8× longer sequence length than the out-of-box DeepSpeed-Ulysses and Megatron-SP.

References

[1] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2022.

[2] FairScale authors. Fairscale: A general purpose modular pytorch library for high performance and large scale training. https://github.com/facebookresearch/fairscale, 2021.

[3] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020.

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

[5] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[6] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

[7] Chiheon Kim, Heungsub Lee, Myungryong Jeong, Woonhyuk Baek, Boogeon Yoon, Ildoo Kim, Sungbin Lim, and Sungwoong Kim. torchgpipe: On-the-fly pipeline parallelism for training giant models. 2020.

[8] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models, 2022.

[9] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[10] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[11] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence parallelism: Long sequence training from system perspective, 2022.

[12] Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7025–7041, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.473.

[13] Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, and Yiran Zhong. Transnormerllm: A faster and better large language model with improved transnormer, 2024.

[14] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020.

[15] 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.

[16] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

[17] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[19] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas

Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

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

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

[22] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[23] 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.

Appendix A. Appendix

A.1. Preliminary

Softmax Attention. Consider the standard attention [20] computation with causal masking in the transformer architecture, formulated as:

$$\mathbf{O} = \mathrm{Softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d} \odot \mathbf{M})\mathbf{V}, \tag{9}$$

where d denotes the hidden dimension. The matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ represent query, key, and value matrices, respectively. These matrices are linear projections of the input $\mathbf{X} \in \mathbb{R}^{N\times d}$, i.e., $\mathbf{Q} = \mathbf{X}\mathbf{W}_\mathbf{Q}$, $\mathbf{K} = \mathbf{X}\mathbf{W}_\mathbf{K}$, $\mathbf{V} = \mathbf{X}\mathbf{W}_\mathbf{V}$. The output matrix is denoted as $\mathbf{O} \in \mathbb{R}^{N\times d}$, and $\mathbf{M} \in \mathbb{R}^{N\times N}$ represents the causal mask matrix. The $\mathrm{Softmax}(\cdot)$ operation introduces quadratic time complexity relative to the input sequence length N, limiting the scalability of vanilla transformers to extended input sequences.

Linear Attention. Linear attention is originally proposed in [5], with the elimination of Softmax operation [20]. Qin et al. [12, 13] propose to replace the Softmax operation with a normalization operation $\mathrm{Norm}(\cdot)$, which turns to a concise formulation as:

$$\mathbf{O} = \mathrm{Norm}((\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V}). \tag{10}$$

When considering bidirectional tasks, the formulation can be simplified as $\mathbf{O} = \mathrm{Norm}((\mathbf{Q}\mathbf{K}^\top)\mathbf{V})$. Then by performing the associativity property of matrix products, it can be mathematically equivalently transformed into a right-product version:

$$\mathbf{O} = \mathrm{Norm}(\mathbf{Q}(\mathbf{K}^\top\mathbf{V})). \tag{11}$$

This linear attention formulation facilitates recurrent prediction with a computational complexity of $O(Nd^2)$. And the recurrent update of $\mathbf{K}^\top\mathbf{V}$ without needing to compute the entire attention matrix makes its inference efficient.

While linear complexity offers significant advantages in terms of computational efficiency and memory optimization for linear attention, it still incurs a proportional increase in computation and memory utilization on a single GPU as the sequence length N grows. This can lead to memory constraints on a single GPU, such as the 80GB limit in NVIDIA A100, for exceptionally long sequences. The challenge of achieving zero-redundancy (on sequence level) training for such long sequences using linear attention-based LLMs across GPU clusters remains an open problem. Furthermore, the complexity of addressing this issue in a casual setting further intensifies the challenge. To address this, we propose LASP as a solution for parallelizing linear attention training at the sequence level, even in a casual setting.

A.2. Communication Analysis

When examining the LASP algorithm, it is important to note that the forward pass requires communication for the $\mathbf{KV}$ activation in each linear attention module layer. The communication volume is determined by Bd^2/h, where B is the batch size and h is the number of heads. In comparison, sequence parallelism in Megatron-LM utilizes all-gather operations twice after two layer normalization layers within each transformer layer, and a reduce-scatter operation after the attention and Feedforward Neural Network (FFN) layers. This results in a communication volume of $2BNd + 4BNd/T$. DeepSpeed uses all-to-all collective communication [17] for input $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, and output $\mathbf{O}$ of each attention module layer, resulting in a communication volume of $4BNd/T$.

Table 1 displays a comparison of communication volumes across three frameworks. d/h is the head dimension which is set at 128 as usual [9]. In practical applications where $N/T \geq 32$, LASP is able to achieve the lowest theoretical communication volume. Furthermore, the communication volume of LASP is not impacted by changes in sequence length N or subsequence length C, which is a huge advantage for extremely long sequence parallelism across large GPU clusters.

Table 1: **Communication Volume Comparison.** Simplified Formulation: we eliminate the common factors Bd for ease of comparison.

Method	Full Formulation	Simplified Formulation
LASP	Bd^2/h	d/h
DeepSpeed-Ulysses	$4BNd/T$	$4N/T$
Megatron-SP	$2BNd + 4BNd/T$	$2N + 4N/T$

A.3. System Engineering Optimization

Kernel Fusion. To improve the efficiency of LASP on GPU, we perform kernel fusion in both the intra-chunk and inter-chunk computations, and also fused the updates of $\mathbf{KV}$ and $\mathbf{dKV}$ into the intra-chunk and inter-chunk computations.

KV State Caching. To avoid recomputing activation $\mathbf{KV}$ during the backward pass, we choose to store it in the HBM of the GPU right after computing it in the forward pass. During the subsequent backward pass, LASP directly accesses $\mathbf{KV}$ for use. It is important to note that the size of the $\mathbf{KV}$ activation cached in HBM is $d \times d$, which is not affected by the sequence length N. When the input sequence length N is exceptionally large, the memory usage of $\mathbf{KV}$ becomes insignificant.

Data Distribution. LASP is designed for training long sequences on linear transformers in a distributed environment, achieved by partitioning the input data along its sequence dimension. In this situation, each GPU within the distributed environment undertakes the training of a subset of sub-sequences, which serves to diminish the large memory footprint associated with activation during the training of long sequences. Communication operations are introduced between GPUs to transmit intermediate states. The final trained model assimilates the knowledge derived from the entirety of the long sequences.

Algorithm 2: LASP Data Distribution

Input: An input sequence in embedding space $\mathbf{X} \in \mathbb{R}^{N \times d}$ with sequence length N and hidden dimension d, distributed world size W and sequence parallel size T;
Obtain number of SP groups $G = W/T$;
Obtain sub-seq length (or chunk size) $C = N/T$;
Get global rank list $R =$ get_global_rank();
Obtain sequence parallel source rank list $R_{src} = \lfloor R/T \rfloor * T$;
Along sequence dimension, split $\mathbf{X}$ into T chunks $\{\mathbf{X}_1, \mathbf{X}_2, ...\mathbf{X}_T\}$, of size $C \times d$ for each;
Transfer copies of data chunks $\{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_T\}$ to GPUs with rank indices in R_{src};
`Scatter` $\{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_T\}$ from R_{src} to all ranks in respective sequence parallel groups.

For an input sequence of length N, we establish its embedding space representation denoted as $\mathbf{X} \in \mathbb{R}^{N \times d}$ with a feature dimension of d. In the LASP framework, $\mathbf{X}$ is evenly partitioned into T chunks, where T is called the sequence parallel size, which must be divisible by the distributed world size W. These segmented data chunks are subsequently assigned to the respective GPUs. It is essential to note that different sequence parallel groups receive dissimilar data batches. However, within the same group, all data chunks originate

from an identical batch of data. A comprehensive depiction of the data distribution process in LASP is provided in Algorithm 2.

Additionally, an illustrative example of data distribution in LASP is presented in Fig. 4, where the distributed world size is characterized by $W = 8$, the sequence parallel size by $T = 4$, the number of sequence parallel groups by $G = 2$, and the sequence parallel source rank list by $R_{src} = [0, 4]$. For the first batch SEQ0, the input sequence $\mathbf{X}$ undergoes partitioning into T chunks $\{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_T\}$ along the sequence dimension, subsequently transmitted to the first rank in SP-GROUP0, which corresponds to global rank 0. The data chunks on global rank 0 are then scattered to global ranks $\{0, 1, 2, 3\}$ within SP-GROUP0, where each rank only retains a single chunk. The subsequent batch SEQ1 follows a similar manner, being assigned to global ranks $\{4, 5, 6, 7\}$ within SP-GROUP1.

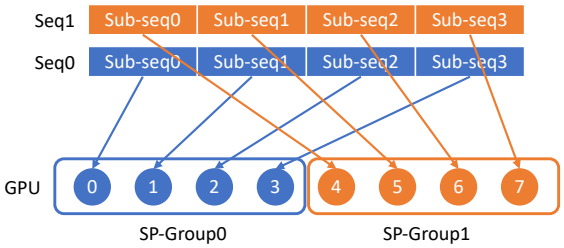

Figure 4: **Example of LASP Data Distribution.**

A.4. Hybrid Parallelism

Data-Sequence Hybrid Parallelism. As explained in Section A.3 and illustrated in Fig. 4, LASP allows for the specification of a smaller sequence parallel size that is divisible by the distributed world size. This configuration results in the input data being split along both the batch and sequence dimensions, which is a type of hybrid parallelism called data-sequence hybrid parallelism. The ZeRO-series optimizers [14] in DeepSpeed and FSDP [23] in PyTorch propose to distribute model states, which include optimizer states, gradients, and model parameters, across all GPUs within the distributed environment. As variants of data parallelism, these techniques seamlessly align with LASP. Furthermore, their focus on minimizing the memory of model states complements LASP's objective of reducing activation memory on each GPU. By integrating these techniques, the training of large models handling long sequence lengths is rendered more practical.

Compatibility with Tensor Parallelism and Pipeline Parallelism. LASP supports both tensor parallelism (TP) and pipeline parallelism (PP). In the context of PP, as exemplified by the GPipe [7] scheduling method, the model is initially partitioned across multiple devices, with each device holding a segment of the model. Data within a mini-batch is then divided into micro-batches, which are sequentially fed into the device containing the first segment. Each device processes its micro-batch and forwards the output to the next device in the sequence, simultaneously preparing to receive and process the subsequent micro-batch from the preceding device. This method of pipelining inputs effectively minimizes device idle times. When LASP is integrated with PP, micro-batches are substituted with sub-sequences derived from a mini-batch. Unlike standard PP, each device retains the intermediate states ($\mathbf{KV}$ in the forward pass and $\mathbf{dKV}$ in the backward pass) locally, rather than transmitting them to the next device as typically done in LASP alone. For TP, the integration with LASP is fluid. Linear attention layers utilize TP to segment matrix operations across both intra-chunk and inter-chunk computations, whereas the handling of MLP layers under TP remains unchanged. The experiment tests on hybrid of LASP, DP, TP and SP will be conducted in the future work.

Table 2: **Convergence Performance of LASP.** All experiments use 8x A100 80G GPUs, sequence length of 16K, and batch size of 1. The results cover various DDP backends in conjunction with LASP. We explore the performance of two linear attention models: TransNormerLLM (TNL) and Linear Transformer, and one transformer model (LLaMA) with Softmax attention, all with 0.4B parameters, across 50K updates.

Model	Parameters	Method	Loss	Method	Loss
Transformer	0.4B	DDP	3.727	\	\
TNL [13]	0.4B	DDP	3.719	LASP + DDP	3.715
		Legacy DDP	3.709	LASP + Legacy DDP	3.705
		FSDP	3.717	LASP + FSDP	3.714
		ZeRO-1	3.653	LASP + ZeRO-1	3.653
		ZeRO-2	3.655	LASP + ZeRO-2	3.649
		ZeRO-3	3.656	LASP + ZeRO-3	3.649
Linear Transformer [5]	0.4B	DDP	5.419	LASP + DDP	5.408
		Legacy DDP	5.425	LASP + Legacy DDP	5.413
		FSDP	5.428	LASP + FSDP	5.441
		ZeRO-1	5.114	LASP + ZeRO-1	5.118
		ZeRO-2	5.105	LASP + ZeRO-2	5.120
		ZeRO-3	5.110	LASP + ZeRO-3	5.123

A.5. Additional Experiment Results

Table 2 presents the convergence results of two linear attention based models: TNL [13] and Linear Transformer [5], and one transformer model (LLaMA [18, 19]) with Softmax attention, evaluated on an epoch-by-epoch basis. The experiments were conducted using the same training corpus: the Pile [3]. Both linear models has 0.4B parameters, demonstrated consistent loss values when training with and without LASP. All experiments undergoes 50K steps. The uniform loss convergence across various DDP backends demonstrates that LASP does not negatively affect model convergence.