APPENDIX

A 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

function STANDALONE_FWD(q, k, v, o, ℓ, m, causal, last)

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.

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.

for $1 \le i \le T_r$ **do**

  Load q_i from HBM to on-chip SRAM.

  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 $o_i^{(0)}$, $\ell_i^{(0)}$, $m_i^{(0)}$.

  for $1 \le j \le T_c$ **do**

    if causal and $i \le j$ **then**

      Continue

    end if

    Load k_j, v_j from HBM to on-chip SRAM.

    On chip, compute $s_i^{(j)} = q_i k_j^T \in \mathbb{R}^{B_r \times B_c}$.

    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), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \text{rowsum}(\tilde{p}_i^{(j)}) \in \mathbb{R}^{B_r}$.

    On chip, compute $o_i^{(j)} = \text{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} o_i^{(j-1)} + \tilde{p}_i^{(j)} v_j^p$.

  end for

  On chip, compute $o_i = \text{diag}(\ell_i^{(T_c)})^{-1} o_i^{(T_c)}$.

  Write o_i to HBM as the i-th block of o.

  if last **then**

    On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.

    Write L_i to HBM as the i-th block of L.

  end if

end for

Return o, ℓ, m and the logsumexp L.

end function

Initialize $\mathbf{O}^p = (0)_{\frac{N}{\mathbb{P}} \times d} \in \mathbb{R}^{\frac{N}{\mathbb{P}} \times d}$, $\ell^{(p)} = (0)_{\frac{N}{\mathbb{P}}} \in \mathbb{R}^{\frac{N}{\mathbb{P}}}$, $m^p = (-\infty)_{\frac{N}{\mathbb{P}}} \in \mathbb{R}^{\frac{N}{\mathbb{P}}}$.

$\mathbf{O}^p, \ell^p, m^p, L^p$ = standalone_fwd($\mathbf{Q}^p, \mathbf{K}^p, \mathbf{V}^p, \mathbf{O}^p, \ell^p, m^p$, True, p=1)

for $1 \le r < p$ **do**

  Receive $\mathbf{K}^r$ and $\mathbf{V}^r$ from **Remote** worker r into HBM.

  $\mathbf{O}^p, \ell^p, m^p, L^p$ = standalone_fwd($\mathbf{Q}^p, \mathbf{K}^y, \mathbf{V}^y, \mathbf{O}^p, \ell^p, m^p$, False, r=(p-1)

  Delete $\mathbf{K}^r$ and $\mathbf{V}^r$ from HBM.

end for

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.

B COMPARISON WITH DEEPSPEED ULYSSES

Method	# GPUs	Sequence Length Per GPU	Sequence Length Total	Time	Speedup
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.