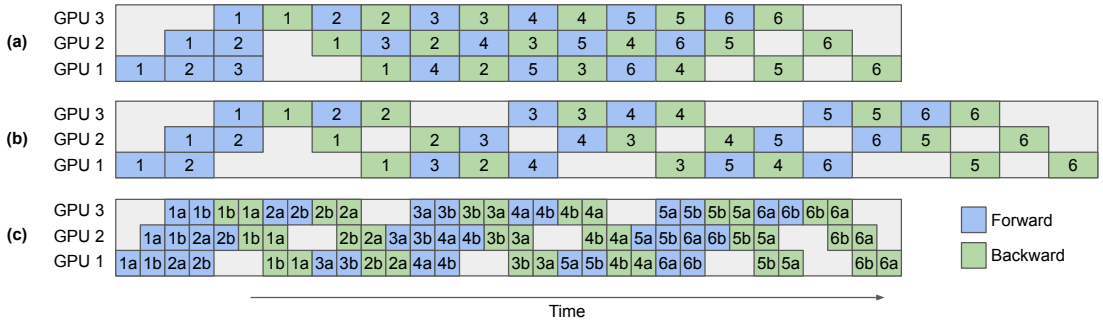
TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models Supplementary Material

A. Combine TeraPipe with Gradient Accumulation

TeraPipe and gradient accumulation (GA) are orthogonal and TeraPipe can further speed up over GA. To see this, we visualize a 3-stage pipeline training with an input batch of 6 training sequences below, similar to Figure 2 in the main paper.



In (a), we show the case where each GPU is capable of storing the intermediate activations of at most 3 input sequences. With scheduling algorithms like DAPPLE (Fan et al., 2020), GA indeed increases the pipeline efficiency. However in (b), when each GPU can only support 2 input sequences (due to large model size), the forward pass of input sequence 3 cannot start on GPU 1 until sequence 1 finishes the backward pass and release the memory of its intermediate activations. The memory constraint limits the pipeline efficiency: only two GPUs can work at a time, and GA cannot solve the issue. In (c), we follow the setting in (b) but enable TeraPipe to split a training sequence into two. TeraPipe improves the pipeline efficiency compared to (b) thanks to more fine-grained pipelining: the three can work at the same time.

In our experiments, we have 48 pipeline stages but a single GPU is only capable to hold 2 input sequences due to its memory capacity. Even with newer GPUs (e.g. 80GB A100, 5x memory compared to V100s in the paper), their memory capacity is still not enough to fulfill the pipeline with 48 input sequences. Therefore, even with GA, TeraPipe is still expected to significantly improve the training efficiency.

B. Implementation

We implement TeraPipe with PyTorch (Paszke et al., 2019) and NCCL (NCCL). We use Megatron-LM (Shoeybi et al., 2019) as the library for operation partitioning and implement microbatch-based pipeline parallelism and data parallelism by ourselves. The core of TeraPipe is implemented using 1714 lines of Python. We include the code in the supplementary material and the code will be open-sourced.

C. Experiment Results

Here, we include the detailed numbers (mean and standard deviation of the latency) and the slicing schemes found by the DP algorithms for all experiments in the main paper. Specifically, we list the details of Figure 5, 6, and 7 in Table 1, 2, and 3.

References

Fan, S., Rong, Y., Meng, C., Cao, Z., Wang, S., Zheng, Z., Wu, C., Long, G., Yang, J., Xia, L., et al. Dapple: A pipelined data parallel approach for training large models. *arXiv preprint arXiv:2007.01045*, 2020.

NCCL. The nvidia collective communication library (nccl). https://developer.nvidia.com/nccl, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Table 1. Detailed numbers and slicing schemes in main experiments (Figure 5 in the main paper).

Model	Setting	Algorithm	Slicing Scheme	Latency (s)	TFlops (per GPU)
GPT3-1B	5, (1)	w/o TeraPipe	[(1, [2048])] * 16	1.517 ± 0.107	0.8841
		w/ TeraPipe	[(1, [776, 640 ,632])] * 16	1.254 ± 0.160	1.0695
	5, (2)	w/o TeraPipe	[(1, [2048])] * 36	1.018 ± 0.065	2.9643
		w/ TeraPipe	[(1, [2048])] * 36	1.018 ± 0.065	2.9643
	5, (3)	w/o TeraPipe	[(1, [2048])] * 72	0.913 ± 0.027	6.6105
		w/ TeraPipe	[(1, [2048])] * 72	0.913 ± 0.027	6.6105
GPT3-13B	5, (4)	w/o TeraPipe	[(1, [2048])] * 16	2.637 ± 0.055	3.0305
		w/ TeraPipe	[(1, [1024, 1024])] * 16	1.891 ± 0.084	4.2261
	5, (5)	w/o TeraPipe	[(1, [2048])] * 32	1.863 ± 0.007	8.5792
		w/ TeraPipe	[(1, [704, 688, 656])] * 32	1.328 ± 0.037	12.0354
GPT3-44B	5, (6)	w/o TeraPipe	[(1, [2048])] * 2	13.319 ± 0.067	0.2148
		w/ TeraPipe	[(1, [64] * 26 + [56] * 6 + [48])] * 2	7.103 ± 0.243	0.4028
	5, (7)	w/o TeraPipe	[(1, [2048])] * 4	4.311 ± 0.032	1.3274
		w/ TeraPipe	[(1, [368, 384, 384, 368, 256, 288])] * 4	2.771 ± 0.112	2.0652
	5, (8)	w/o TeraPipe	[(1, [2048])] * 8	2.662 ± 0.001	4.2995
		w/ TeraPipe	[(1, [384, 384, 368, 320, 296, 296])] * 8	1.111 ± 0.002	10.3018
GPT3-175B	5, (9)	w/o TeraPipe	[(1, [2048])] * 2	9.990 ± 0.005	1.1300
		w/ TeraPipe	[(1, [120] * 4 + [112] * 6 + [104] * 8 + [64])] * 2	1.481 ± 0.002	7.6225
	5, (10)	w/o TeraPipe	[(1, [2048])] * 2	5.822 ± 0.003	1.9390
		w/ TeraPipe	[(1, [128] * 16)] * 2	1.160 ± 0.001	9.7318

Table 2. Detailed numbers and slicing schemes in ablation studies on the effectiveness of the dynamic programming algorithm (Figure 6 in the main paper).

Model	Setting	Algorithm	Slicing Scheme	Latency (s)	TFlops (per GPU)
GPT3-44B	6, (a)	#Slices=1	[(1, [2048])] * 8	2.662 ± 0.001	4.2995
		#Slices=4	[(1, [512] * 4)] * 8	1.241 ± 0.003	9.2226
		#Slices=8	[(1, [256] * 8)] * 8	1.255 ± 0.004	9.1197
		#Slices=16	[(1, [128] * 16)] * 8	1.241 ± 0.003	9.2226
		DP	[(1, [384, 384, 368, 320, 296, 296])] * 8	1.111 ± 0.002	10.3018
GPT3-175B	6, (b)	#Slices=1	[(1, [2048])] * 2	9.990 ± 0.005	1.1300
		#Slices=4	[(1, [512] * 4)] * 2	2.902 ± 0.003	3.8900
		#Slices=8	[(1, [256] * 8)] * 2	1.892 ± 0.002	5.9667
		#Slices=16	[(1, [128] * 16)] * 2	1.547 ± 0.01	7.2973
		#Slices=32	[(1, [64] * 32)] * 2	1.593 ± 0.002	7.0866
		#Slices=64	[(1, [32] * 64)] * 2	2.227 ± 0.002	5.0691
		#Slices=128	[(1, [16] * 128)] * 2	3.252 ± 0.004	3.4714
		DP	[(1, [120] * 4 + [112] * 6 + [104] * 8 + [64])] * 2	1.481 ± 0.002	7.6225

Table 3. Detailed numbers and slicing schemes in experiments with longer sequence lengths (Figure 7 in the main paper).

Model	Input Sequence Length	Algorithm	Slicing Scheme	Latency (s)	TFlops (per GPU)
GPT3-13B	2048	w/o TeraPipe	[(1, [2048])] * 32	1.863 ± 0.007	8.5792
		w/ TeraPipe	[(1, [704, 688, 656])] * 32	1.328 ± 0.037	12.0354
	4096	w/o TeraPipe	[(1, [4096])] * 8	2.526 ± 0.001	1.5819
		w/ TeraPipe	[(1, [552, 536, 528, 512, 504, 496, 488, 480])] * 8	0.913 ± 0.085	4.3765
	6144	w/o TeraPipe	[(1, [6144])] * 4	3.754 ± 0.006	0.5322
		w/ TeraPipe	[(1, [584, 568] + [512] * 6 + [496, 488, 472, 464])] * 4	0.756 ± 0.008	2.6427
	8192	w/o TeraPipe	[(1, [8192])] * 2	4.978 ± 0.004	0.2007
		w/ TeraPipe	[(1, [512] * 6 + [480] * 2 + [416] * 10)] * 2	0.636 ± 0.001	1.5707