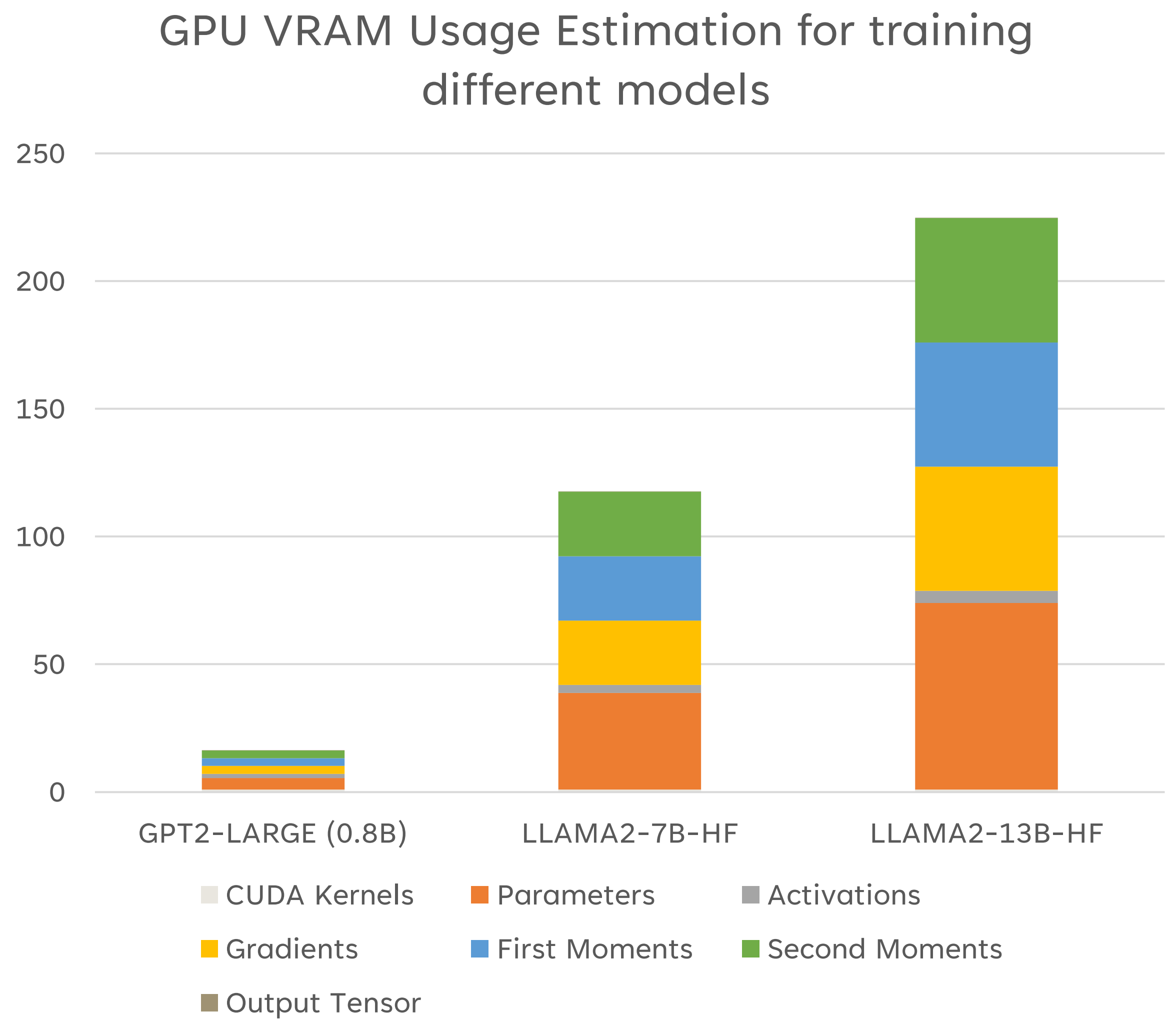
MEGATRON-LM

TRAINING MULTI-BILLION PARAMETER LANGUAGE MODELS USING MODEL PARALLELISM

Mohammad Shoeybi, Mostofa Patwary, Raul Puri,
Patrick LeGresley, Jared Casper, Bryan Catanzaro

Large Models Require Large Memory

- Training >> Inference
- Adam > SGD
- Larger minibatch
- More parameters
- …
- More devices are needed to:
    - Speed up training
    - Simply enable training

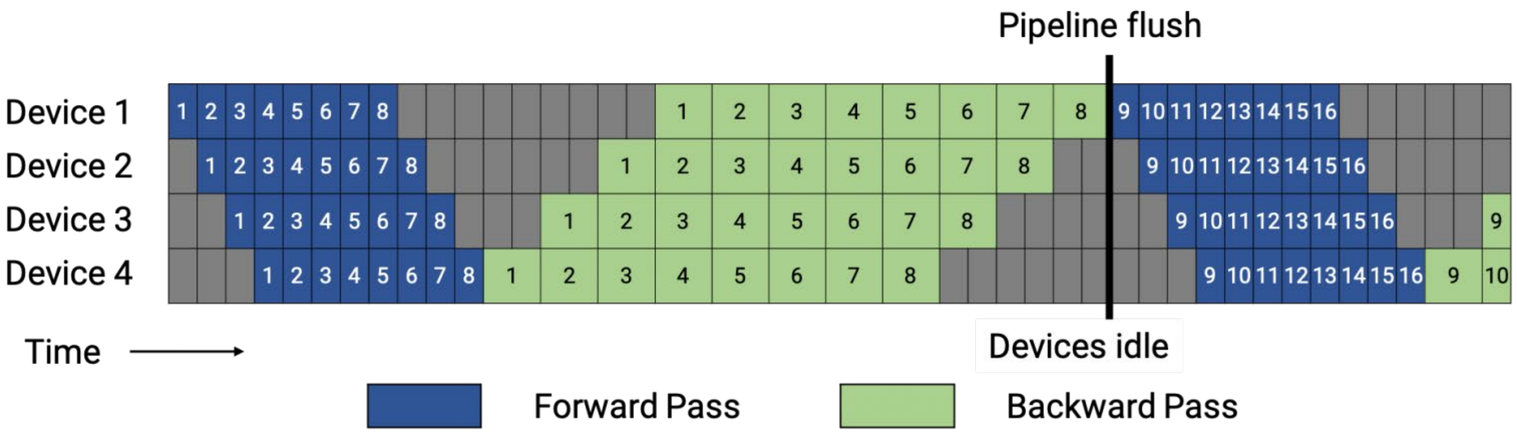


Data Parallelism

- Partitions a training minibatch across multiple devices
- Linearly scalable
- Slicing activation only
- Does not help with excessive model size

Pipeline Parallelism

- Layer-wise model parallelism
- Bubbles reduce efficiency
- Requires additional logic to handle pipelining

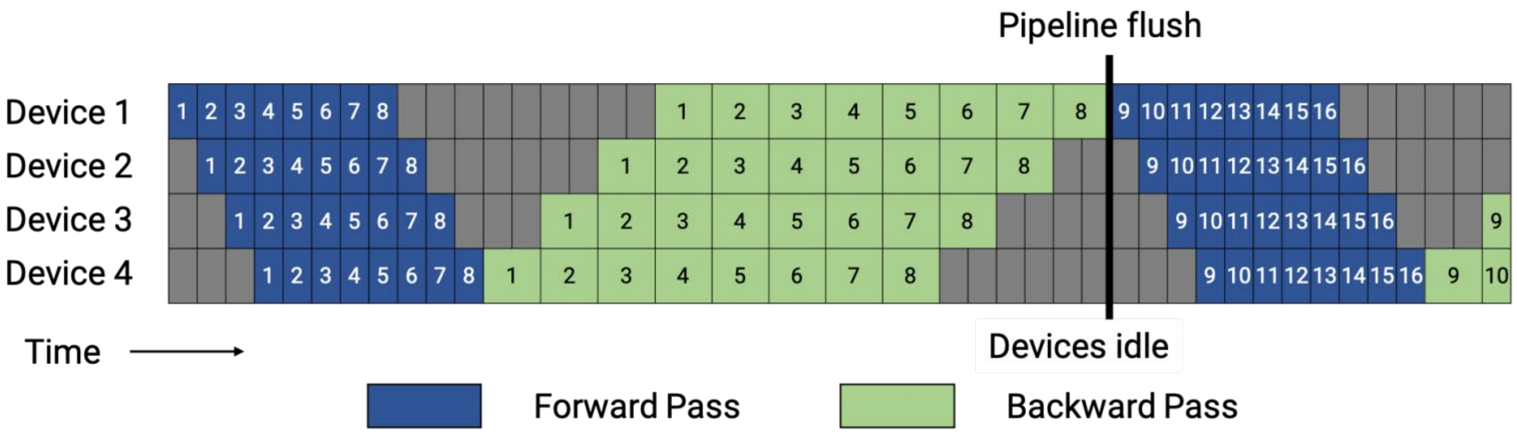


Pipeline schedule of Gpipe*

* Image from https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

Pipeline Parallelism

- Bubbles reduce efficiency
  - Larger batch sizes for relatively small bubbles are impractical
  - Low scaling efficiency



Pipeline schedule of Gpipe*

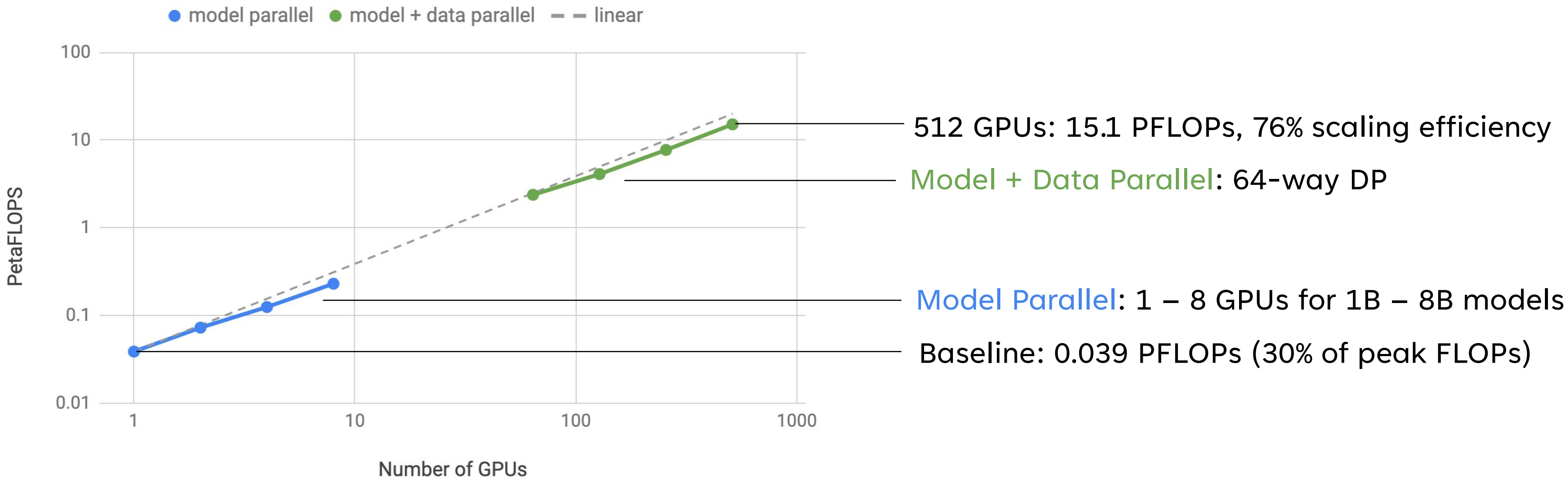
$$\text{bubble time fraction} = \frac{p-1}{m}$$

p: pipeline parallelism $= 4$
m: micro batch size $= 8$

* Image from https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

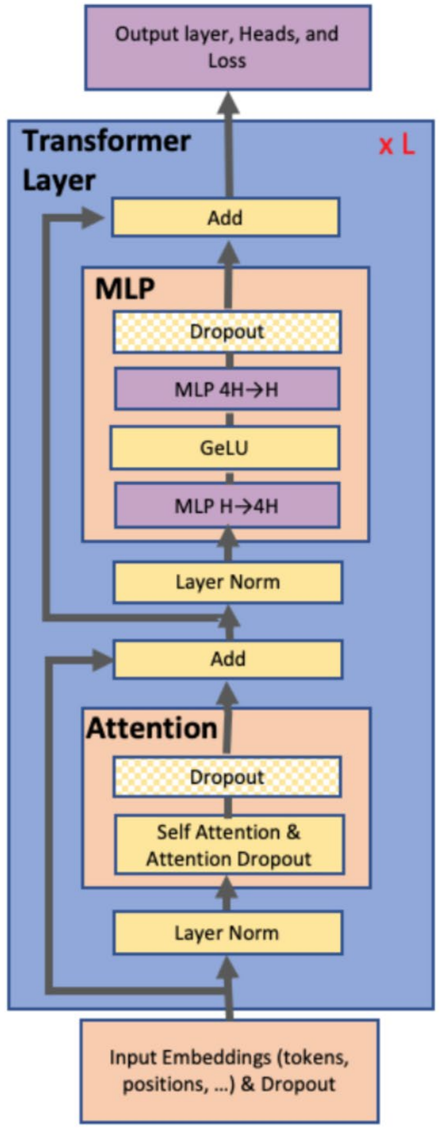
Tensor Parallelism

- Intra-layer model-parallelism
- Good scaling efficiency inside one node
- Orthogonal to DP and PP: combine to get best strategy



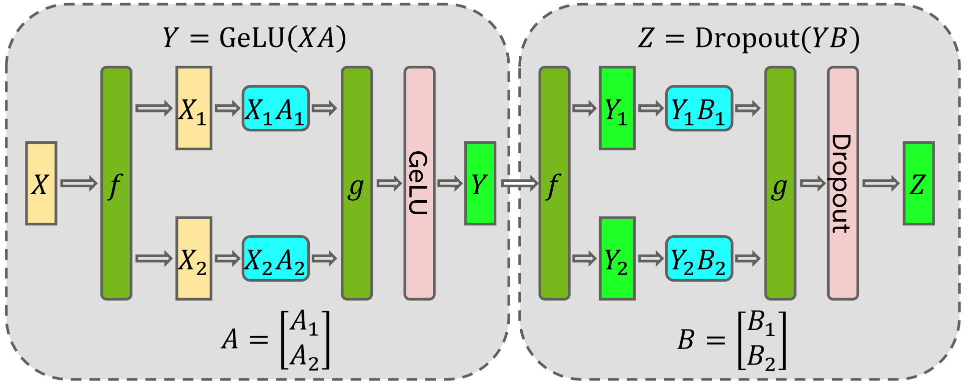
Tensor Parallelism

- Tailored for transformer networks
- Transformer Layer
  - Self-attention block
  - Two-layer MLP
- Targets:
  - Exploit parallelism wherever possible
  - Reduce synchronization cost



Tensor Parallelism: MLP

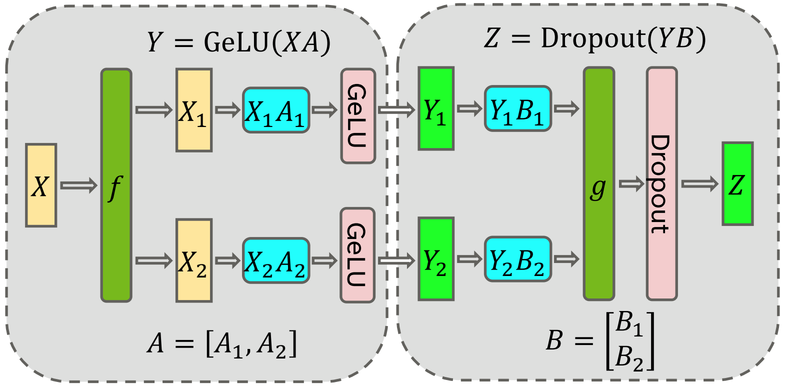
- Two layers: $Y = \mathrm{GeLU}(XA)$, $Z = \mathrm{Dropout}(YB)$
- Option 1: Split A and B along their rows
  - $X = [X_1, X_2], A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$
  - $Y = \mathrm{GeLU}(X_1A_1 + X_2A_2) \neq \mathrm{GeLU}(X_1A_1) + \mathrm{GeLU}(X_2A_2)$
  - Synchronizations before GeLU and Dropout



f: SPLIT (forward)
ALL_GATHER (backward)
g: ALL_REDUCE (forward)
IDENTITY (backward)

Tensor Parallelism: MLP

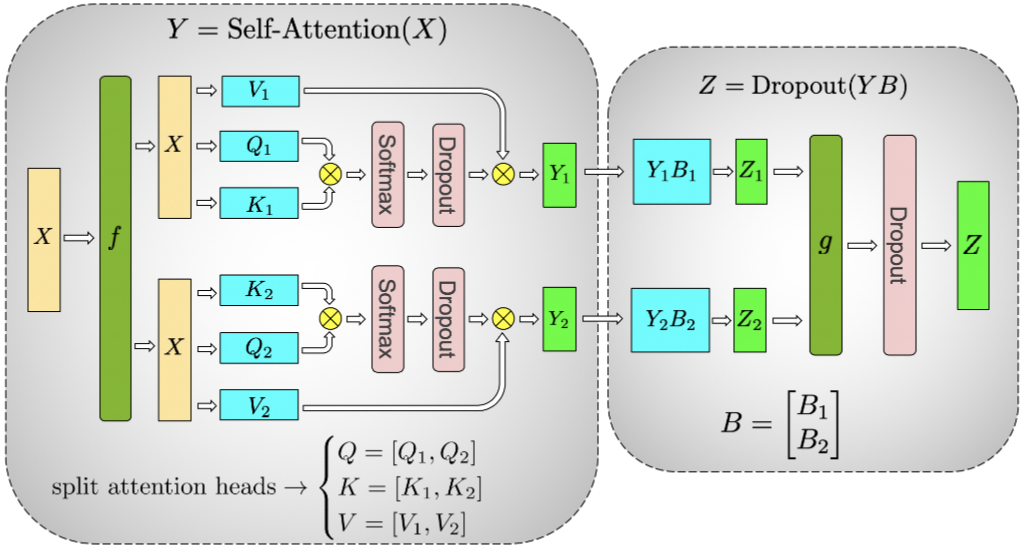
- Two layers: $Y = \mathrm{GeLU}(XA), Z = \mathrm{Dropout}(YB)$
- Option 2: Split A along its columns
  - $A = [A_1, A_2]$
  - $Y = [Y_1, Y_2] = \mathrm{GeLU}([XA_1, XA_2]) = [\mathrm{GeLU}(XA_1), \mathrm{GeLU}(XA_2)]$
  - Split B along its rows
  - Only one sync needed before $\mathrm{Dropout}$



f: IDENTITY (forward)
ALL_REDUCE (backward)
g: ALL_REDUCE (forward)
IDENTITY (backward)

Tensor Parallelism: Self-Attention

- Attention block: Self-attention followed by a linear layer
  - Partition K, Q, V along their columns
  - Slice the subsequent MM weight along its rows
  - Only one sync required



f: IDENTITY (forward)
ALL_REDUCE (backward)
g: ALL_REDUCE (forward)
IDENTITY (backward)

Tensor Parallelism: Communication cost

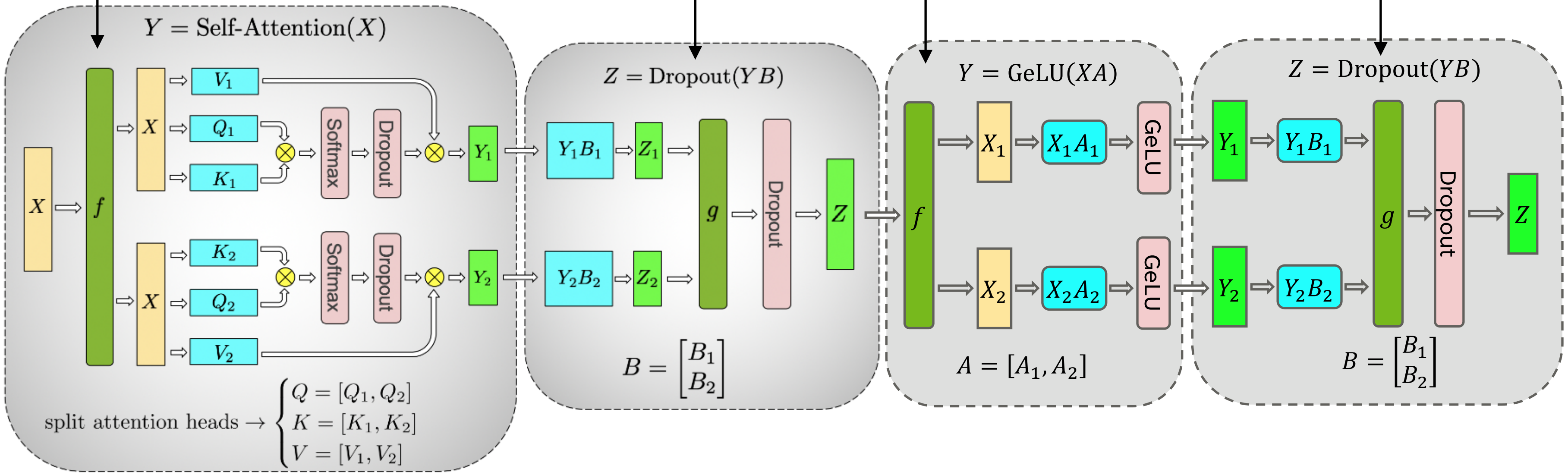
- 4 Total communication operations in 1 forward + backward pass

ALL_REDUCE at backward

ALL_REDUCE at forward

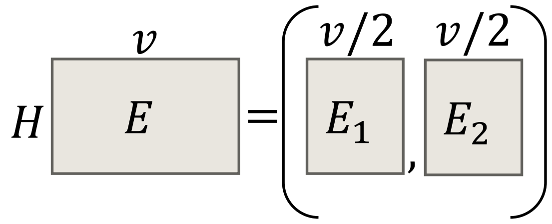
ALL_REDUCE at backward

ALL_REDUCE at forward

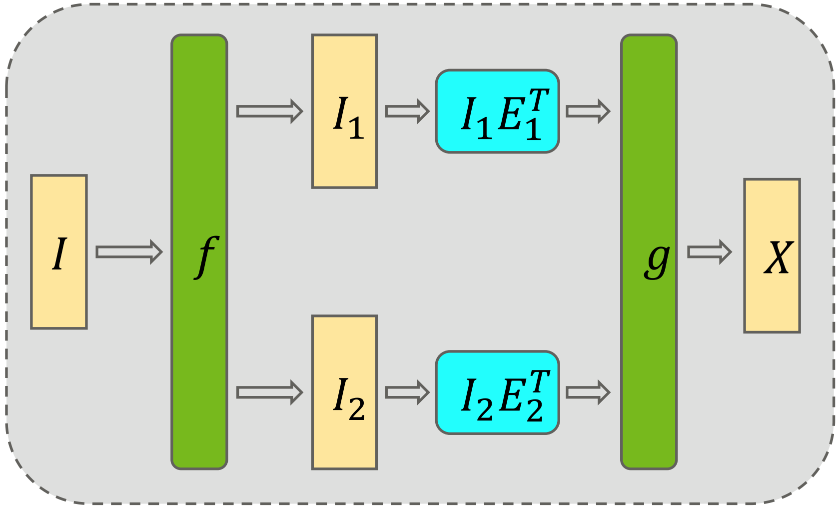


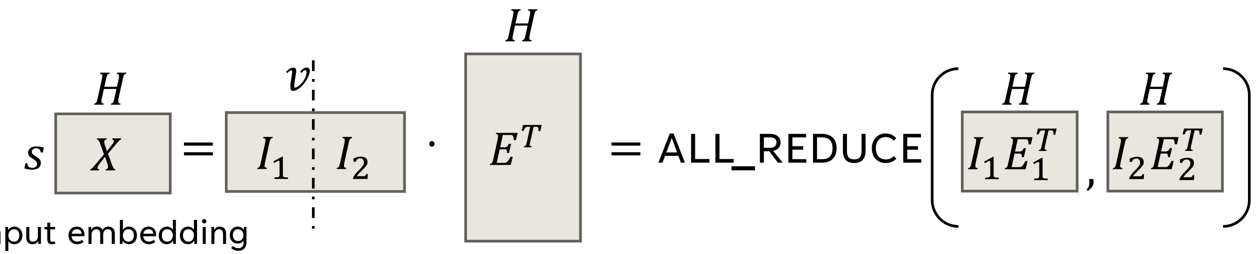
Tensor Parallelism: I/O embedding

- Embedding matrix $E_{H \times v}$: Hidden-size × $v$ocab-size*
  - Weights shared between input and output embeddings
- Parallelized along columns (vocab dimension)
- Input embedding: acquired using ALL_REDUCE



f: SPLIT (forward)
ALL_GATHER (backward)
g: ALL_REDUCE (forward)
IDENTITY (backward)





Input embedding

Tensor Parallelism: I/O embedding

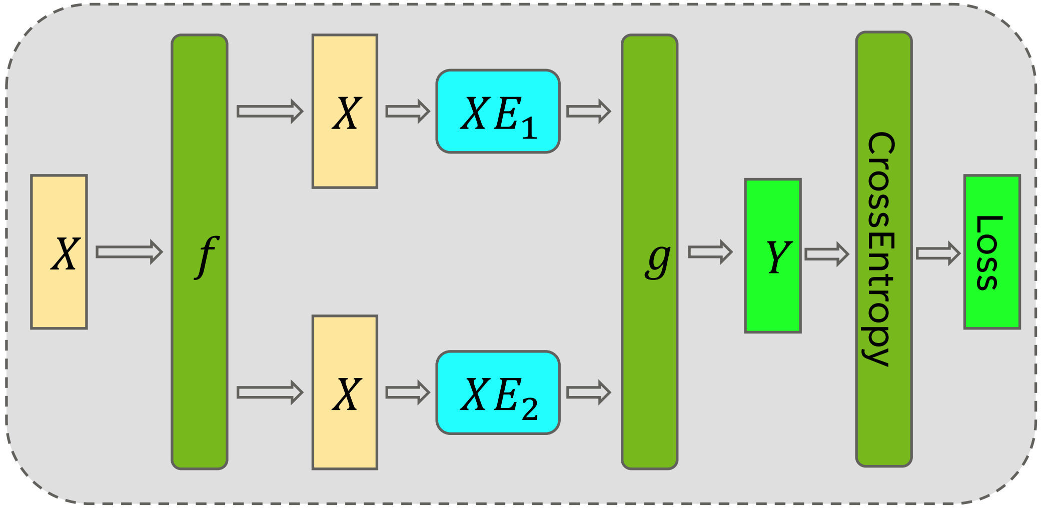
- Output embedding: acquired using ALL_GATHER
  - Costly: This will communicate $b \times s \times v$ elements
    - b: batch size
    - s: sequence length

$$\underset{H}{\overset{v}{E}} = \left(\overset{v/2}{E_1}, \overset{v/2}{E_2} \right)$$

f: IDENTITY (forward)
ALL_REDUCE (backward)
g: ALL_GATHER (forward)
SPLIT (backward)

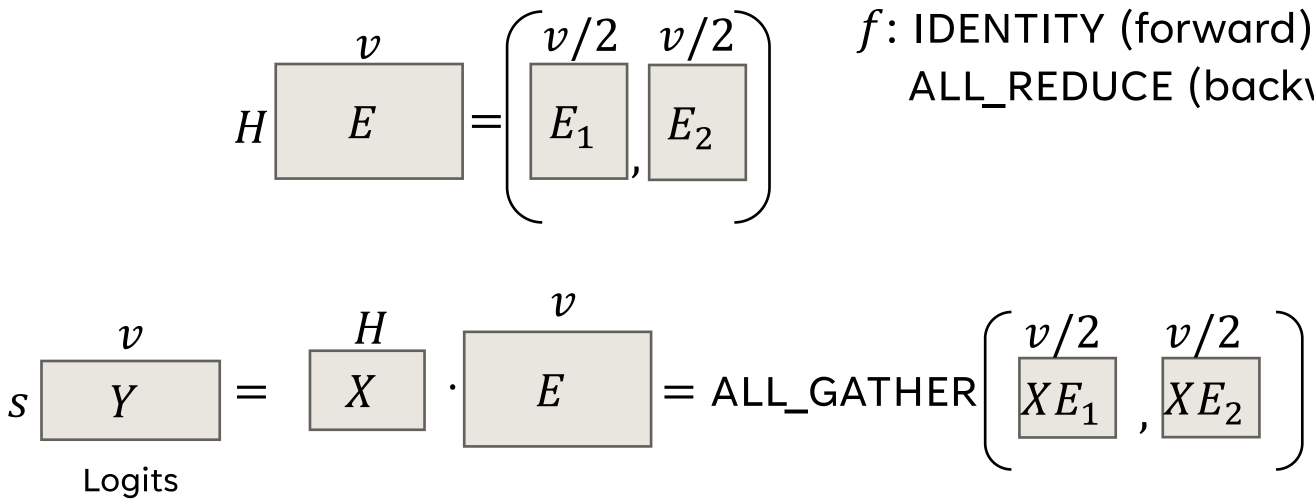
$$\underset{s}{\overset{v}{Y}} = \overset{H}{X} \cdot \overset{v}{E} = \text{ALL\_GATHER}\left(\overset{v/2}{XE_1}, \overset{v/2}{XE_2} \right)$$

Logits

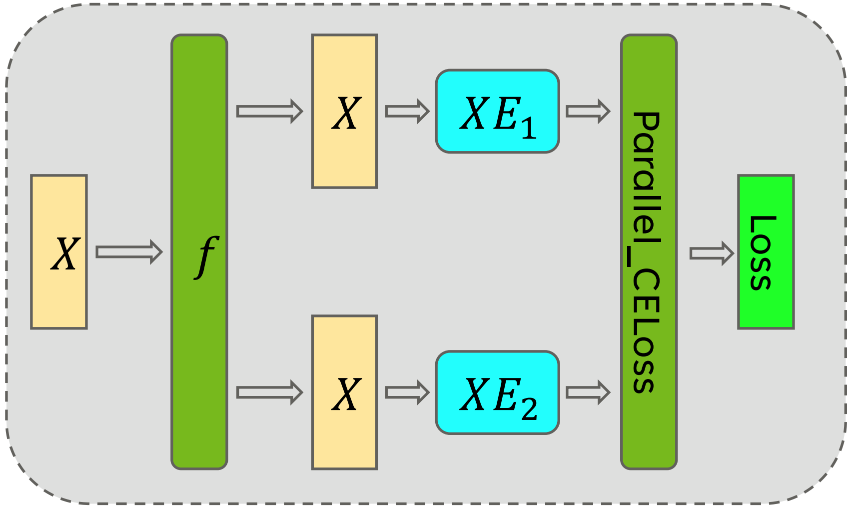


Tensor Parallelism: I/O embedding

- Output embedding: acquired using ALL_GATHER
  - Fuse the output $[Y_1, Y_2]$ with the cross-entropy loss
  - Communication reduced to $b \times s$



f: IDENTITY (forward)
ALL_REDUCE (backward)



Experiments

- 32 DGX-2H servers
  - 512 V100 SXM3 32GB GPUs
- Intra-server connection
  - 300 GB/s NVSwitch
- Inter-server connection
  - 100 GB/s InfiniBand (8 per server)

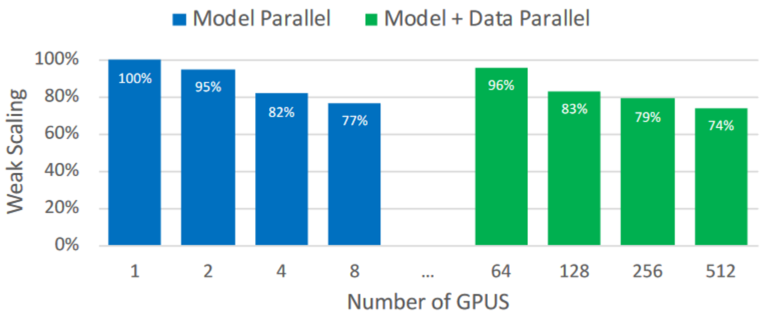
- GPT-2 & BERT Models
- TP (+ DP)
- Mixture of datasets

Experiments: Scalability

- GPT-2: 1B – 8B
- Hidden size: 96
- Parameters/GPU: ~1B

- Weak Scaling @ 512 GPUs: 74%

Hidden Size	Attention heads	Number of layers	Number of parameters (billions)	Model parallel GPUs	Model +data parallel GPUs
1536	16	40	1.2	1	64
1920	20	54	2.5	2	128
2304	24	64	4.2	4	256
3072	32	72	8.3	8	512

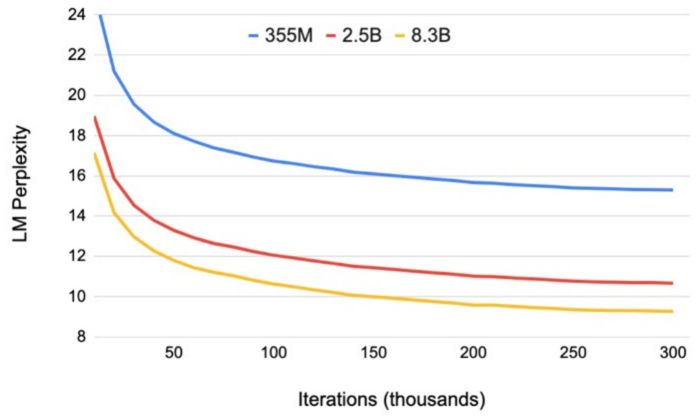


Experiments: GPT-2

- 0.4B, 2.5B, 8.3B

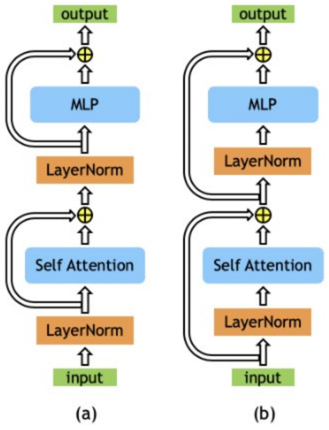
Parameter Count	Layers	Hidden Size	Attn Heads	Hidden Size per Head	Total GPUs	Time per Epoch (days)
355M	24	1024	16	64	64	0.86
2.5B	54	1920	20	96	128	2.27
8.3B	72	3072	24	128	512	2.10

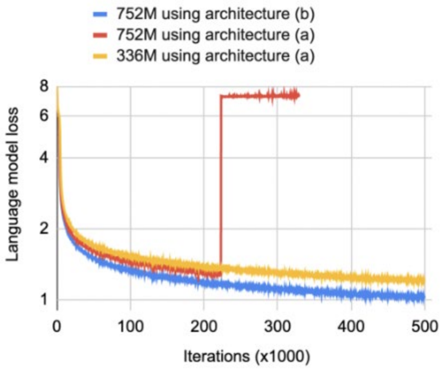
Model	Wikitext103 Perplexity ↓	LAMBADA Accuracy ↑
355M	19.31	45.18%
2.5B	12.76	61.73%
8.3B	**10.81**	**66.51%**
Previous SOTA	15.79	63.24%



Experiments: BERT

- 0.3B, 1.3B, 3.9B
- Modified architecture from (a) to (b) to allow for larger models

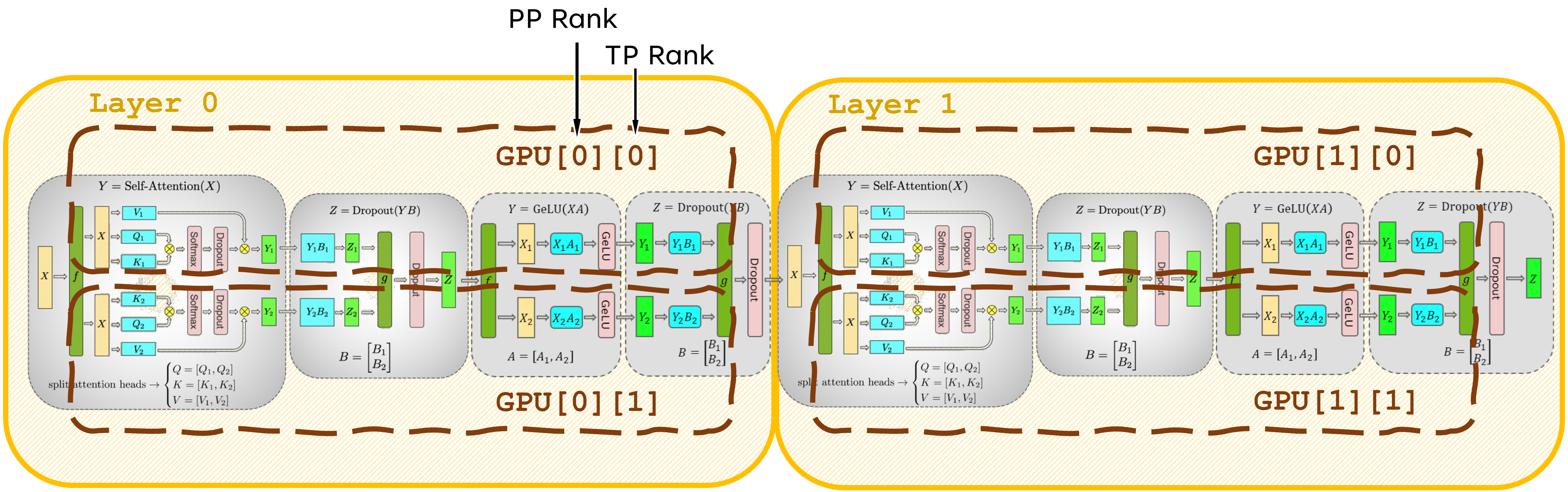




Model	trained tokens ratio	MNLI m/mm accuracy (dev set)	QQP accuracy (dev set)	SQuAD 1.1 F1 / EM (dev set)	SQuAD 2.0 F1 / EM (dev set)	RACE m/h accuracy (test set)
RoBERTa (Liu et al., 2019b)	2	90.2 / 90.2	92.2	94.6 / 88.9	89.4 / 86.5	83.2 (86.5 / 81.8)
ALBERT (Lan et al., 2019)	3	90.8	92.2	94.8 / 89.3	90.2 / 87.4	86.5 (89.0 / 85.5)
XLNet (Yang et al., 2019)	2	90.8 / 90.8	92.3	95.1 / 89.7	90.6 / 87.9	85.4 (88.6 / 84.0)
Megatron-336M	1	89.7 / 90.0	92.3	94.2 / 88.0	88.1 / 84.8	83.0 (86.9 / 81.5)
Megatron-1.3B	1	90.9 / 91.0	92.6	94.9 / 89.1	90.2 / 87.1	87.3 (90.4 / 86.1)
Megatron-3.9B	1	**91.4 / 91.4**	**92.7**	**95.5 / 90.0**	**91.2 / 88.5**	**89.5 (91.8 / 88.6)**
ALBERT ensemble (Lan et al., 2019)				95.5 / 90.1	91.4 / 88.9	89.4 (91.2 / 88.6)
Megatron-3.9B ensemble				**95.8 / 90.5**	**91.7 / 89.0**	**90.9 (93.1 / 90.0)**

Combining Multiple Parallelism Techniques

- Model_Parallelism = Tensor_Parallelism × Pipeline_Parallelism



Combining Multiple Parallelism Techniques

- Best practice can be chosen from multiple combinations

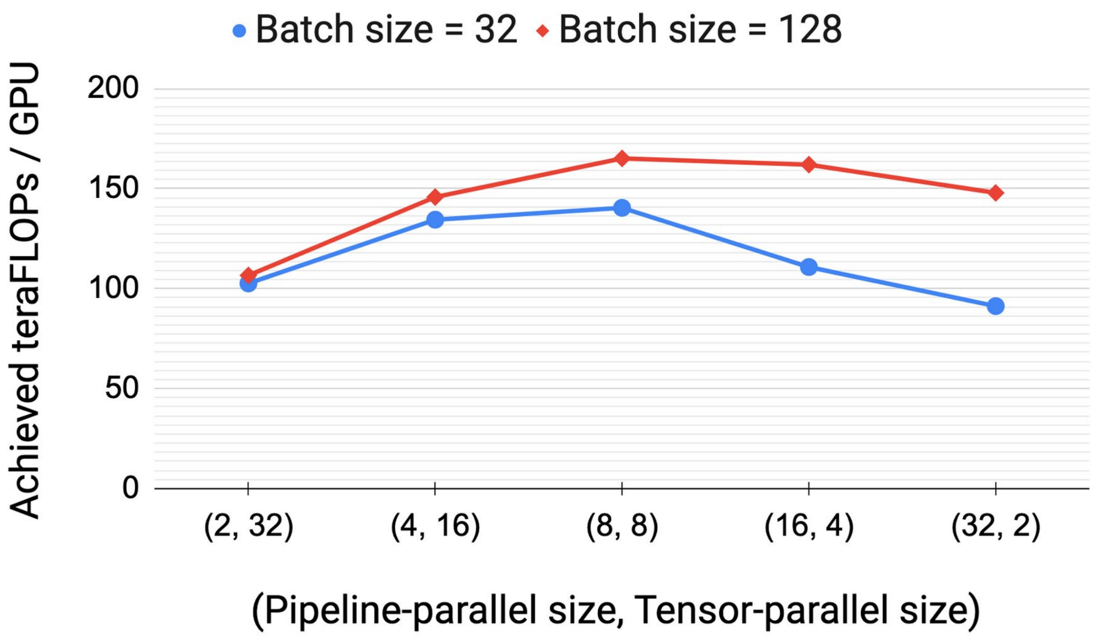
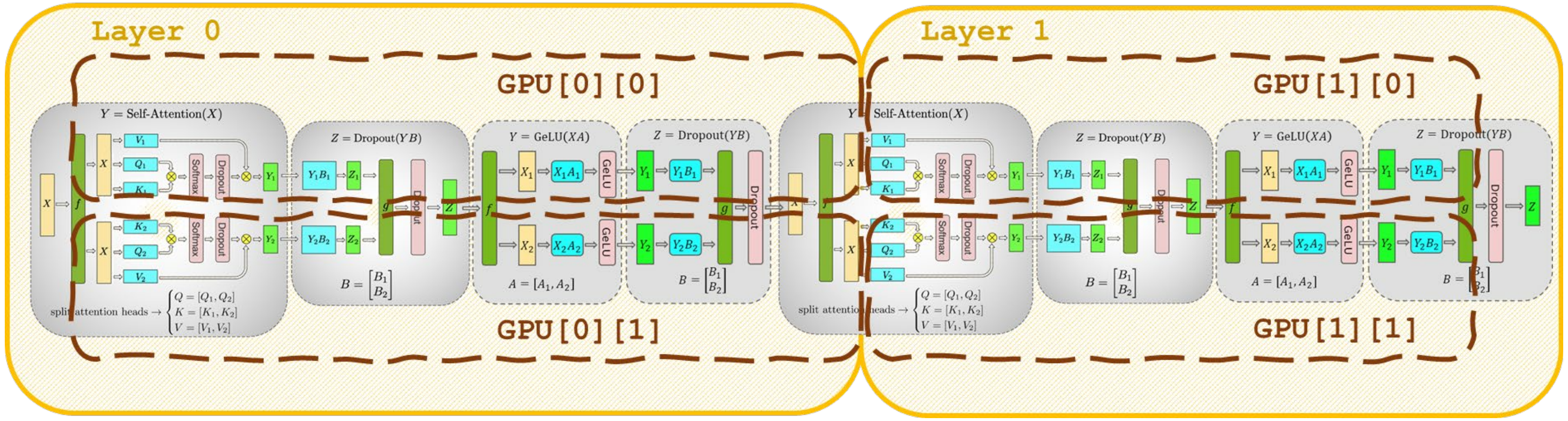


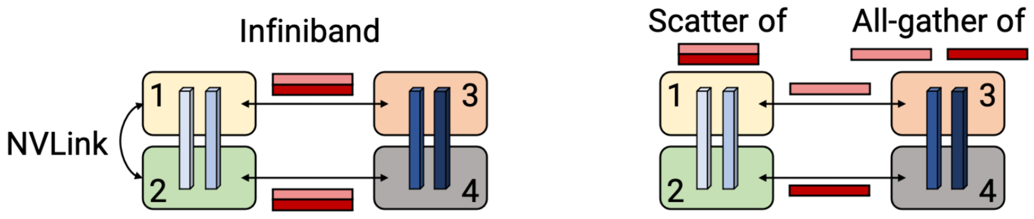
Figure 9. Throughput per GPU of various parallel configurations that combine pipeline and tensor model parallelism using a GPT model with 162.2 billion parameters, two different batch sizes, and 64 A100 GPUs.

* Image from https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

Combining Multiple Parallelism Techniques

- Further reduce communication cost between parallel layers





(a) W/o scatter/gather optimization. (b) W/ scatter/gather optimization. *

* Image from https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

Combining Multiple Parallelism Techniques

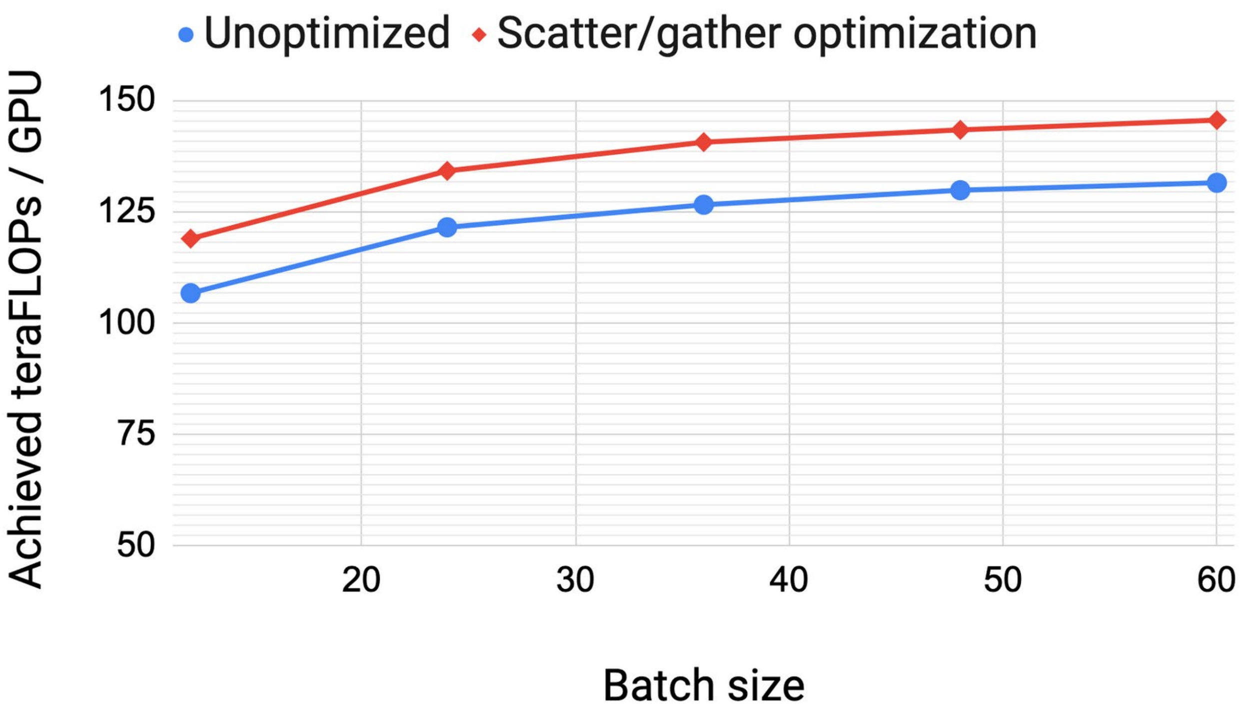
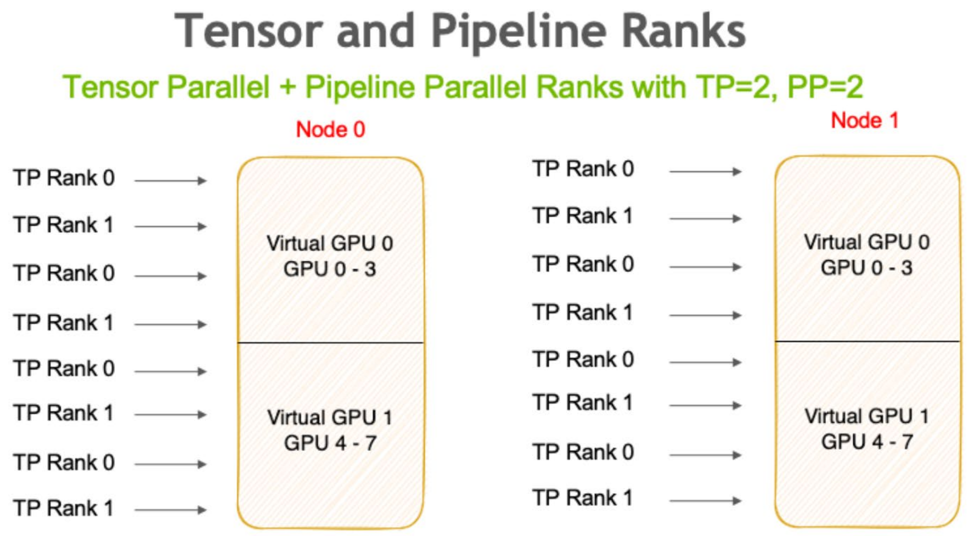


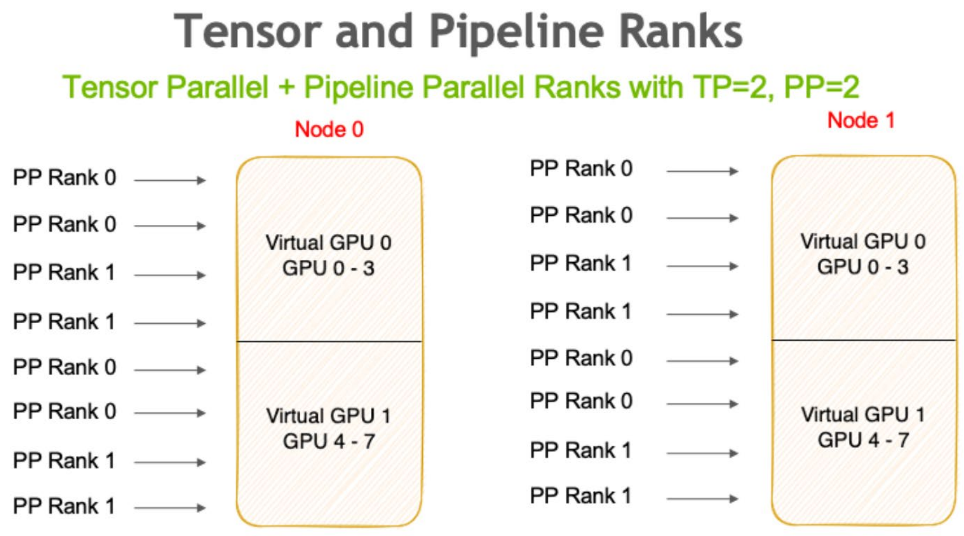
Figure 10. Throughput per GPU with and without the scatter/gather optimization for a GPT model with 175 billion parameters using 96 A100 GPUs and the interleaved schedule.

* Image from https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

Combining Multiple Parallelism Techniques

- (MP=TP×PP) GPUs = 1 Virtual GPU



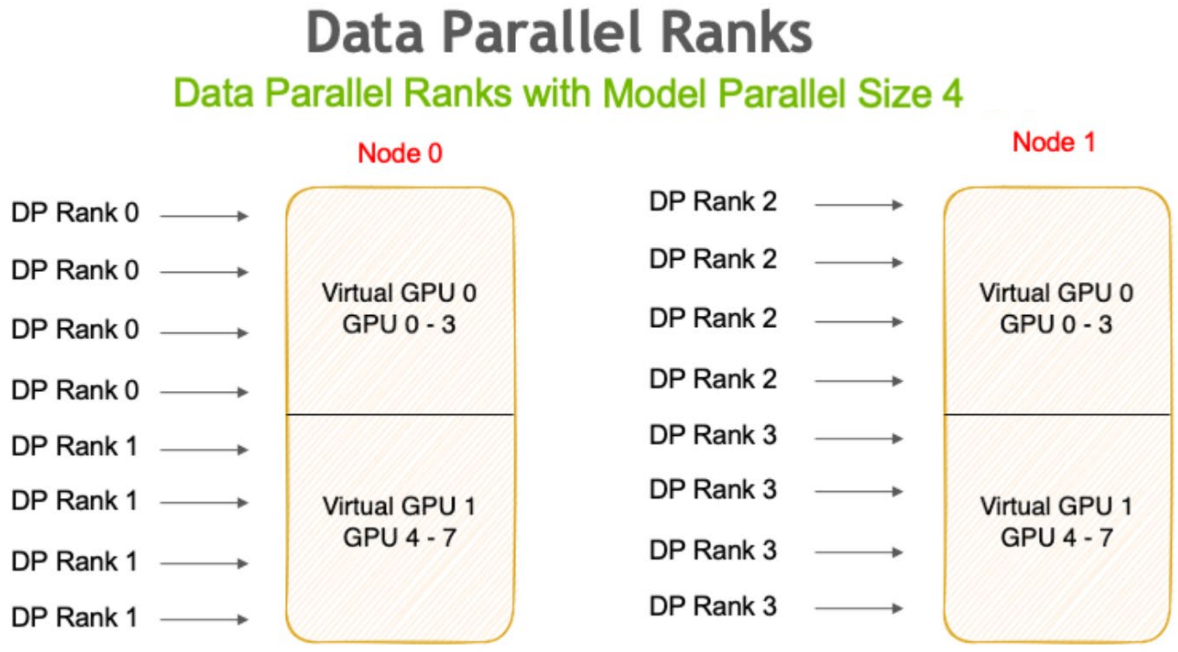


* Images from https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/main/nlp/nemo_megatron/parallelisms.html

Megatron-LM: Training Multi-billion Parameter Language Models Using Model Parallelism

Combining Multiple Parallelism Techniques

- DP_max = (# of GPUs) / (# of Virtual GPUs)



* Image modified from https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/main/nlp/nemo_megatron/parallelisms.html

Discussion

- Efficient model parallelism with good scalability
- Compatible with other parallelism methods
- Very useful codebase (https://github.com/NVIDIA/Megatron-LM)

- TP brings large communication cost
  - The limit of efficient TP depends largely on # of GPUs on a single node
  - Speed will be significantly affected if intra-node connection is bad
  - One slow GPU will make the entire system much slower
  - Dependencies prevent interleaving of communication and computation[1]

[1] Singh, S. et al. Communication-minimizing Asynchronous Tensor Parallelism. https://arxiv.org/pdf/2305.13525.pdf

THANK YOU!

Presented by Yufeng Du